LOW-LEVEL SOFTWARE FOR AN EHSI DEVELOPMENT SYSTEM

by

DAVE GRUENBACHER

B.S., Kansas State University, 1985

A MASTER'S THESIS

submitted in partial fulfillment

of the requirements for the degree

MASTER OF SCIENCE

Department of Electrical Engineering

KANSAS STATE UNIVERSITY

Manhattan, Kansas

1987

Approved by:

Stephen A. Dyer

Major Professor

TABLE OF CONTENTS

TABLE OF CONTENTS (cont.)

LIST OF FIGURES

LIST OF FIGURES (cont.)

I. INTRODUCTION

1.1 History and Purpose of the EHSI Development System

Ever since the Wright brothers made their historic first flight, man has been trying to make flying an easier and safer task. Advances in aeronautical engineering continue to make better aircraft that are easier to fly. But, today, more advancements are being made in the cockpit area. A multitude of analog and digital signals are presented in the cockpit of any aircraft. For many years, all the pilot saw of the signals was an analog meter or a lamp. Recently, though, the age of electronics has made possible methods of displaying flight data on cathode-ray tube (CRT) screens. These displays can greatly reduce pilot workload, thus increasing the safety of flying.

An electronic horizontal situation indicator (EHSI) is a system that displays flight-related data on a cockpit display screen. EHSI's have been developed for military aircraft and commercial airlines, and they have proven to be very useful to the pilots.

Because of the expense of the systems, the general aviation community has not been offered a practical EHSI. With the recent advances in microprocessor technology and the decrease of costs associated with electronics, the possibility of building an affordable EHSI for the general aviation community is being explored.

S. A. Dyer [1] has proposed that an EHSI be developed for general-aviation at a reasonable cost. The EHSI would consist of one or two flight-capable high-resolution screens, a command keyboard, and the necessary electronics and software to display the flight data in useful forms. Display pages showing inflight, navigation, and instrument landing data are proposed. The display pages developed concurrently with this report will be discussed in Chapter 7.

An EHSI Development System is being built at Kansas State University (KSU) to look into the possibilities of producing an EHSI in the eight-to twelve-thousand dollar range. The targeted consumer is the general-aviation community.

1.2 Elements of the EHSI Development System

The basic tasks of the EHSI system include: (1) sampling the analog flight signals, (2) assimilating information to be displayed, and finally, (3) sending the information in a suitable graphical form to the display device. It would also be convenient to include a keypad so that the pilot can interact with the system. Fig. 1 shows a block diagram of the EHSI development system at KSU. An interface driven by a Motorola 68000 based board has been designed and built by J. Lagerberg [2]. This interface, known as the Data Acquisition and Communications Interface

(DACI), controls communication within the system. It also digitizes flight data for use by the rest of the system. The interface detects key presses of the command keyboard on an interrupt basis, and also sends graphics commands to the Hewlett-Packard 1345A vector graphics display. The 1345A is a $4,000 dollar unit with high resolution and is specified for operation at pressure altitudes up to 15,000 feet. The "Vector Graphics Memory" (VGM) option of the HP-1345A eliminates the need for constant refresh of the screen from the DACI. In other words, the 1345A display is refreshed from commands in the VGM. If the display needs to be changed, the DACI can change parts of the memory, thus altering the display. The flight data being digitized is coming from an ATC-610 Flight Simulator. The ATC-610 provides analog information similar to that available during flight in an airplane. Therefore, the possibility exists of developing near "real-life" situations in the laboratory for the EHSI Development System.

The host computer, a Zenith-158 personal computer, acts as the central processing unit for the EHSI system. The Z-158 does all of the display generation and calculation work for the system. The Z-158 is connected to the DACI via the parallel printer port. Communication has been established between the DACI and the Z-158 in an interrupt environment supported by a handshaking protocol. The routines to accomplish the interface are written in 8088 assembly-

language. Assembly-language is used for maximum speed, but an intermediate-level language, C, is used for the display generation routines. Therefore, interfacing the assembly-language routines with the C functions has also been accomplished.

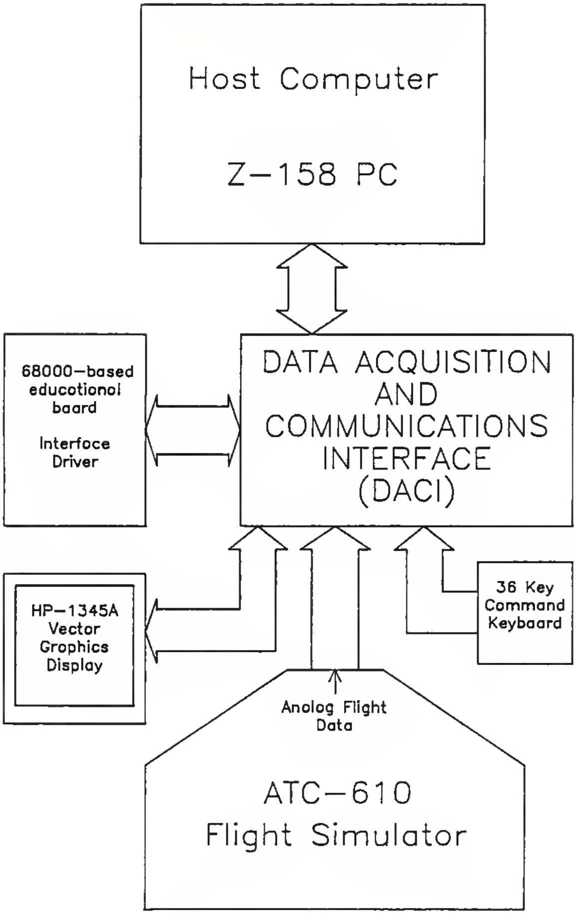

Figure 1. Block diagram of EHSI Development System.

1.3 Research Goals and Purpose of This Report

The research covered in this thesis was directed toward the development of the required low-level software to create a suitable operating environment for the Z-158. The goals of the research are:

1. To develop algorithms and implement them in assembly language routines to establish communications between the Z-158 and DACI. The routines must be reliable, easy to access, and fast enough to operate the EHSI in near real-time.
2. To interface the Z-158 with commands written for the DACI by Lagerberg [2].
3. To interface the communications routines with the C functions.
4. To provide reliable and easy to use guidelines for using the communications routines.
5. To analyze samples of data taken from the flight simulator and design digital filters to provide lowpass filtering and differentiation of the analog signals.
6. To provide adequate maintenance information and recommendations to future project members.

II. SPECIFICATIONS

2.1 Communications are carried out through the parallel port

The Z-158 parallel port is a 25-line port that is internally divided into three different addressable ports. One address is for an 8-line I/O databus. Another is for input control lines, and the last is for output control lines. One of the lines can be used to trigger hardware interrupts. The lines used for the Z-158 -to- DACI interface are shown in Fig. 2. Actual hardware connections can be found in Fig. 10.

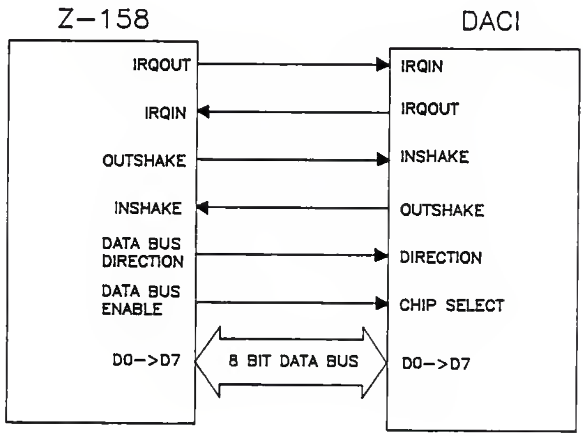

Figure 2. Control and Data lines for the Z-158 -to- DACI interface.

The Z-158 and DACI each have an IRQOUT, IRQIN, OUTSHAKE, and INSHAKE line.

IRQOUT is the line used to initiate an interrupt. This line is connected to the receiving end's IRQIN line. The Z-158 uses its IRQOUT line to interrupt the DACI, and the DACI uses its IRQOUT line to interrupt the Z-158.

The IRQIN lines of the Z-158 and DACI are latched by hardware so that interrupts will not be missed. The Z-158 IRQIN line is connected internally to its 8259A programmable interrupt controller. The DACI IRQIN line is latched to a 6821 PIA. Additional information on the interrupt hardware design of the DACI is available in [2].

All external hardware interrupts are tied to the 8259A within the Z-158. The controller prioritizes interrupts and tells the 8088 central processing unit (CPU) within the Z-158 what interrupt should be serviced next. The 8259A can be programmed to ignore certain interrupts and give higher priority to others. A discussion of how the 8259A is programmed is presented in Sec. 3.5.

DATA BUS ENABLE and DATA BUS DIRECTION are two lines that the Z-158 uses to control the data bus buffer (74LS245) found on the DACI. Before attempting a read or a write to the DACI, the Z-158 must make sure the data bus is appropriately set up. When the data bus is not in use, the bus is kept in the high-impedance state.

The data bus has 8 lines and is bidirectional. A

modification was made to the Z-158 parallel port so that the data bus could be read. The modification is shown and explained in Appendix C.

2.2 Communications Between the Z-158 and DACI

DACI interrupting the Z-158

The DACI interrupts the Z-158 on the occurrence of one of several physical events. The DACI pulses the IRQOUT line after placing the appropriate interrupt vector on the data bus. The Z-158 must then suspend its current process and initiate an interrupt service routine. The Z-158 retrieves the interrupt vector from the data bus and then pulses the OUTSHAKE line. The interrupt service routine of the Z-158 concludes and the suspended process continues. The DACI continues after seeing the acknowledge pulse on its INSHAKE line. The timing diagram in Fig. 3 shows the states of the pertinent lines.

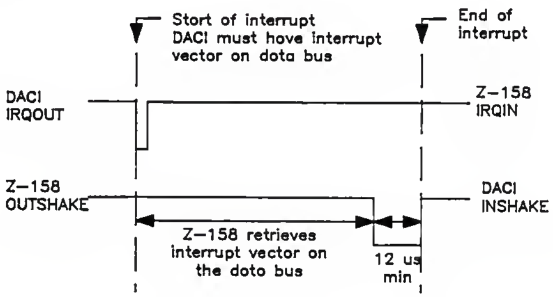

Figure 3. Timing diagram of DACI interrupting the Z-158.

The physical events that cause the DACI to interrupt the Z-158 are:

event	interrupt vector sent
System switch turned on	65H
System switch turned off	66H
System clock updated	60H
A key has been pressed	01H-->23H*

* interrupt vector is the number of the key pressed. See Fig. 30.

Z-158 interrupting the DACI

The Z-158 interrupts the DACI by pulsing its IRQOUT line after placing an interrupt command vector on the data bus. The four commands and their respective command vectors are given below:

COMMAND NUMBER	FUNCTION
01H:	The Z-158 requests that the flight data package be sent. The data package contains current digitized values of the flight simulator signals, and the system clock.
02H:	The Z-158 wants to send screen commands to the HP-1345A. The words are transferred until the top byte of a screen command contains FFH.
03H:	The Z-158 requests that a certain area of HP-1345A vector memory commands be sent back to the Z-158.
04H:	The Z-158 tells the DACI to toggle the current state of the alarm.

Each command will now be explained in detail.

Command 01H:

The Z-158 places 01H on the data bus and pulses its IRQOUT line. The DACI will acknowledge by pulsing its OUTSHAKE line. To set up for the transfer of the data package, the DACI places the number of entries of the data package on the data bus and pulses its OUTSHAKE line. The Z-158 reads the number and acknowledges with a pulse on its OUTSHAKE line. The transfer of the data package will then occur within a handshaking environment. After the final entry is received and acknowledged, the Z-158 and DACI return to their interrupted processes.

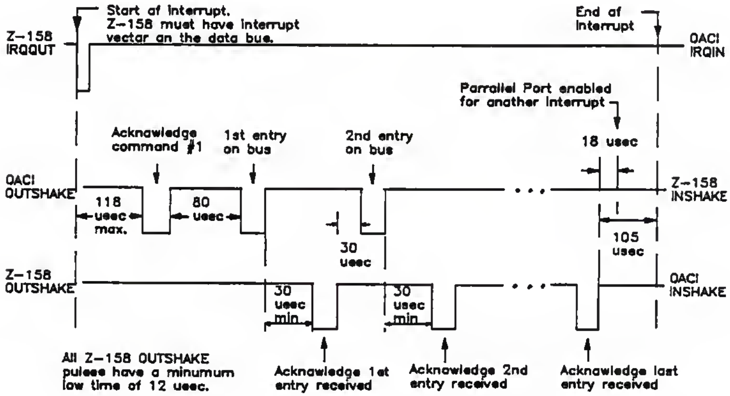

Figure 4. Interface Command 1 timing diagram.

Command 02H:

The Z-158 places 02H on the data bus and pulses its IRQOUT line. The DACI acknowledges on its OUTSHAKE line,

and the transfer of screen words is ready to begin. Each word is sent as two bytes, first the most-significant byte (MSB) and then the least-significant byte (LSB). The Z-158 places the MSB on the data bus and strobes the DACI. After the DACI reads the byte and acknowledges on its OUTSHAKE line, the Z-158 places the LSB on the data bus and strobes the DACI. The screen words are sent in this fashion until the MSB of a screen word is FFH. This signals the end of the transfer, and the Z-158 and the DACI return to their interrupted processes.

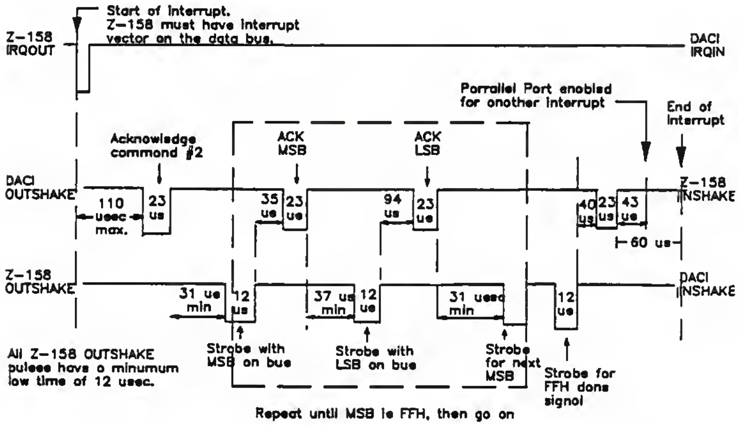

Figure 5. Interface Command 2 timing diagram.

Command 03H:

The Z-158 places 03H on the data bus and pulses its IRQOUT line. The DACI acknowledges on its OUTSHAKE line. Using the same handshaking protocol as used in the previous

2 commands, the Z-158 sends the starting address and ending address of the screen memory desired. The DACI then proceeds to transfer the requested screen words to the Z-158. After the last word is sent, the DACI and Z-158 return to their interrupted processes.

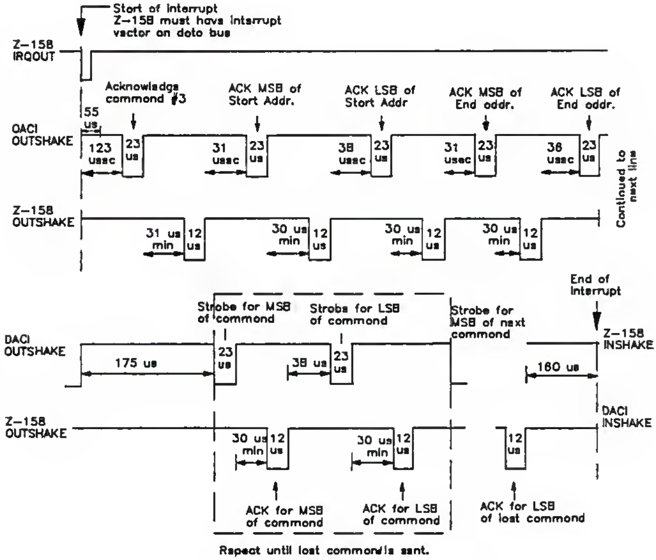

Figure 6. Interface Command 3 timing diagram.

Command 04H:

The Z-158 places 04H on the data bus and pulses its IRQOUT line. The DACI will acknowledge the interrupt on its

OUTSHAKE line. The DACI will proceed to toggle the state of the alarm, and the Z-158 and DACI will return to their interrupted processes.

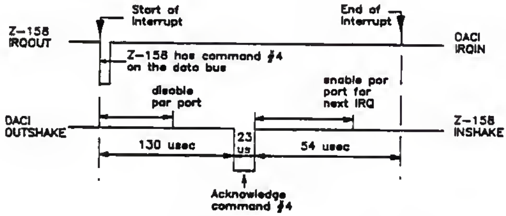

Figure 7. Interface Command 4 timing diagram.

2.3 Software Choices

It is worth mentioning the two languages used to implement the algorithms for the interrupt procedures.

Microsoft Macro Assembler (MASM) Version 4.0 by Microsoft, Inc. was used to write the 8088 assembly language routines -- the communication routines which involve interrupt servicing and handshaking. The use of MASM allows good speed of execution and tight control over the object code.

Microsoft C Version 4.0 was used as the high-level language because of its portability, speed compared to other high-level languages, bit manipulation functions, and the extensive libraries that are included with the compiler.

Because the assembler and the compiler were produced by the same company, interfacing the two proved to be a fairly straightforward task.

III. THE LOW LEVEL SOFTWARE INTERFACE

3.1 Preview of EHSI Operating Environment

Before detailing the communication routines to implement the commands described in Chapter 2, a basic description of how the routines are used is needed. The main program of the system initializes the system, acts on interrupts received from the DACI, and restores the system on shutdown.

The first thing the main program does when invoked is install the interrupt handler and prepare the environment for the EHSI system. The main program will then poll the first element of the "Interrupt Vector Stack", waiting for a non-zero vector. The interrupt handler is the only module that can place vectors on the interrupt vector stack. It places interrupt vectors on the stack when the DACI interrupts the Z-158 with a vector on the data bus. If the main program is slow in servicing the vectors on the stack, the interrupt handler has the capability of stacking the interrupts so that all of them are serviced. As the main program services the interrupts, it rolls the stack down if it detects more than one interrupt on the interrupt vector stack.

When the main program detects an interrupt vector it enters a decision loop to determine what kind of action to take. If a screen needs to be updated, Command 1 will be

called to fetch the current data package. Calculation of the HP-1345A screen code will then take place. Finally, Command 2 will be called to send the screen code through the DACI to the HP 1345A.

The control of the program would then return back to polling the Interrupt Vector Stack. When the shutdown vector is received, a function to restore the environment to its original state is called. The main program would then terminate. See Fig. 8 for an illustration showing system interaction.

Now that the basics of the system have been covered, the low level software can be developed in detail. Sections 3.2 and 3.3 will develop tools for the routines, and Sections 3.4 and 3.5 will show the actual development of the routines.

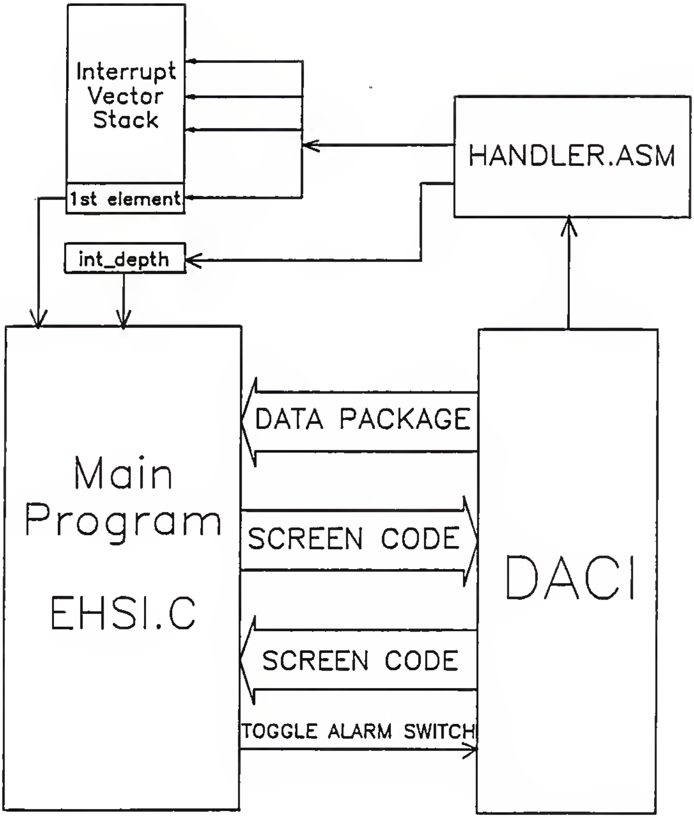

Figure 8. System interaction.

3.2 Communicating with the hardware

Before developing the interface functions, it is useful to describe the 8088 assembly language instructions used to communicate through the ports.

To input a byte from a port with address PORT_ADDRESS, the following assembly language code is required:

```
mov     dx, PORT_ADDRESS            ; put port number in dx
in      al, dx                      ; input byte is now in al
```

The byte read from port PORT_ADDRESS is now in the low byte of accumulator ax, otherwise known as al.

To output a byte, OUTPUT_BYTE, through a port with address PORT_ADDRESS, the following assembly language code is required:

```
mov     dx, PORT_ADDRESS            ; put port number in dx
mov     al, OUTPUT_BYTE             ; put output byte in al
out     dx, al                      ; output al to port dx
```

Three ports control all the lines used in the parallel port interface from the Z-158 to the DACI. Port 378H is the data bus port. Reading this port will read the data bus, and outputting a byte to this port will put that byte on the data bus. Port 379H is the input control line port. Only two lines of Port 379H are used, D4 and D6. D4 is the INSHAKE line, or where the DACI OUTSHAKE's to. D6 is the IRQIN line, or where the DACI IRQOUT's to. Port 37AH is the output control line port. Five of the lines are used. D0 is the Z-158 OUTSHAKE line, D2 is the Z-158 IRQOUT line, D1 controls the direction of the data bus, and D3 controls the

state of the data bus. D4 is set high to enable parallel port interrupts, and low to disable IRQ7.

Port number	line(s)	Z-158 function	DACI connection
378H	DO->D7	Data Bus	Data Bus
379H	D4	INSHAKE	OUTSHAKE
379H	D6	IRQIN	IRQOUT
37AH	DO*	OUTSHAKE	INSHAKE
37AH	D1*	DATA BUS DIRECTION	DIRECTION
37AH	D2	IRQOUT	IRQIN
37AH	D3*	DATA BUS ENABLE	CHIP SELECT
37AH	D4	IRQ7 ENABLE	NONE

*Actual outputs of these lines are inverted.

Figure 9. Z-158 parallel port line connections.

Figure 10 is a schematic showing the hardware interface in more detail. Notice the three Z-158 ports and their respective names. With the port and line functions now identified, commands can be written to perform specific actions.

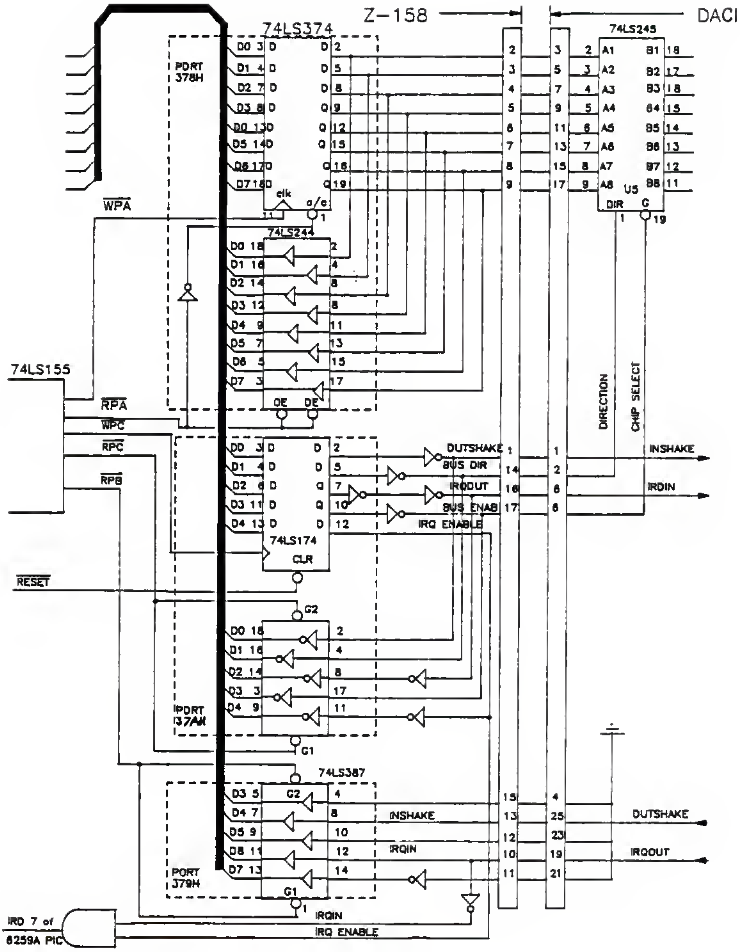

Figure 10. Z-158 -to- DACI Hardware Interface.

The different states of the databus control lines, DATA_BUS_ENABLE and DATA_BUS_DIRECTION, are summarized in Fig. 11. It should be noted that these lines are inverted in the Z-158.

ACTION	DATA_BUS_DIRECTION PORT 37AH Line D3	DATA_BUS_ENABLE PORT 37AH Line D3
Enable Z-158 to Read	Low	Low
Enable Z-158 to Write	High	Low
Disable Data Bus	Low*	High

*This would normally be a "Don't Care", but keeping the Z-158 in read mode will help to avoid bus conflicts.

Figure 11. Control states of the data bus control lines.

Line D4 of Port 37AH is the IRQ_ENABLE line. IRQ_ENABLE must be high before a high-low transition on the IRQ_IN line can trigger an interrupt. (See Fig. 10.)

Noting the inverters between Port 37AH's output latch and the 25-pin connector, the states of the port can be found to accomplish different tasks.

To prepare to read the data bus, the following assembly language instructions are issued:

```
mov dx, 037AH
mov al, 0FEH
out dx, al
```

Following the data bus preparation is the actual read:

```
mov dx, 0378H
in  al, dx
```

After the read, it is good practice to disable the data bus. Since the input byte is in al, it should be stored before performing the following disable bus procedure:

```
mov dx, 037AH
mov al, 0F4H
out dx, al
```

To perform a write to the data bus, a similar procedure of enabling, writing, and disabling the data bus is used:

Enable data bus for a write:

```
mov dx, 037AH
mov al, 0FCH
out dx, al
```

Output the byte OUTPUT_BYTE:

```
mov dx, 0378H
mov al, OUTPUT_BYTE
out dx, al
```

Disable the data bus:

```
mov dx, 037AH
mov al, 0F4H
out dx, al
```

Other useful sequences of code that are used frequently in handshaking are:

To clear the OUTSHAKE line:

```
mov dx, 037AH          ; load port address
in  al, dx             ; get current state
or  al, 01H            ; set bit D0
out dx, al             ; output the byte
```

Similarly, to set the OUTSHAKE line:

```
mov dx, 037AH          ; load port address
in  al, dx             ; get current state
or  al, 0FEH           ; clear bit D0
out dx, al             ; output the byte
```

To check the level of the INSHAKE line:

```
        mov dx, 0379H        ; load port address
        in  al, dx           ; get current state
        and al, 10H          ; result if 0 if INSHAKE low
                             ; result is 1 if INSHAKE high
```

All of the procedures written so far have been for communications through the parallel printer port, but two ports internal to the Z-158 are important for interrupt handling. Both ports, 20H and 21H, talk directly with the 8259A Programmable Interrupt Controller. The 8259A prioritizes external hardware interrupts, such as the keyboard interrupt and system clock interrupt. The parallel port interrupt is wired directly to the 8259A.

Port 21H gives access to the interrupt mask register. The mask controls which interrupts are serviced or ignored. A low level in the mask corresponds to interrupts being serviced, and a high level means that interrupt is disabled. The table in Fig. 12 shows the interrupts connected to the 8259A and the interrupt mask register bit corresponding to each [4].

DOS Interrupt No.	Function	8259A Interrupt Register	Mask Bit
08H	System Clock	Port 21H	D0
09H	Keyboard Interrupt	Port 21H	D1
0AH	Not used	Port 21H	D2
0BH	Asynch. Port 1	Port 21H	D3
0CH	Asynch. Port 2	Port 21H	D4
0DH	Fixed Disk Controller	Port 21H	D5
0EH	Floppy Disk Controller	Port 21H	D6
0FH	Parallel Port Interrupt	Port 21H	D7

Figure 12. Interrupts vectored through the 8259A.

To enable and disable interrupts vectored through the 8259A, read Port 21H, set or clear the desired mask bits, and send the new mask back to Port 21H. The assembly code can be written as:

```
        mov dx, 21H                 ; port number of mask
        in  al, dx                  ; read current mask
(set or clear desired bits)         ; change the mask
        out dx, al                  ; write new mask
```

When interrupts are funnelled through the 8259A, a special command is given at the end of the interrupt service routine to clear the 8259A. Port 20H is the control port used for this purpose. After an interrupt, the following assembly code is used to clear the 8259A:

```
        mov dx, 20H                 ; control port number
        mov al, 20H                 ; End of Interrupt byte
        out dx, al                  ; send EOI to 8259A
```

3.3 Interfacing Assembly Language and C

One last subject will now be covered before developing the communication routines between the Z-158 and DACI. The assembly-language routines must be able to read and write to data variables and structures in the main C routine. Writing these routines thus requires some knowledge of the architecture of the 8088's memory.

Memory on the 8088 processor is divided into segments of up to 64K each. Segments are given names to correspond to their contents, and similar segments are grouped together when modules are linked together. Since Microsoft products

were used, Microsoft's segment model was adhered to in the structure of the interface. The segments and their relative locations in memory are shown in Fig. 13.

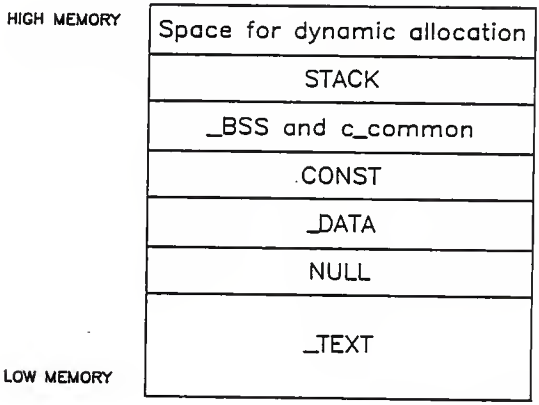

Figure 13. Microsoft segment model.

The contents of the segments shown in Fig. 13 are listed below:

SEGMENT	CONTENTS
STACK	The STACK segment contains the user's stack. This will be a common stack to the C and assembly language routines.
_BSS	The _BSS segment contains all uninitialized static data items.
c_common	The c_common segment contains all global uninitialized data items.

CONST — The **CONST** segment contains all constants that can only be read.

_DATA — The **_DATA** segment is the default data segment. All initialized global and static data are put in this segment.

NULL — The **NULL** segment is a special purpose segment that is placed at the bottom of DGROUP. The segment contains the compiler copyright notice, and is checked before and after program execution.

_TEXT — The **_TEXT** segment contains all the code of the C and assembly language routines.

The above definitions are for small-model programs. A small-model program has two segments that contain all the segments listed above. One segment contains all the code, and the other contains the data. Maximum size for each is 64K, so maximum size of the program cannot exceed 128K. If a problem with size ever becomes a problem, a medium-model program could then be created.

The _TEXT and _DATA segments will be the ones referred to when implementing an assembly-language routine to be called from a C program. The code for the assembly-language routine should be placed in the _TEXT segment, and global data must be placed in the _DATA segment, or else the C and assembly-language will not be able to access the same data. The _DATA segment was defined before to contain all initialized global data items, so data structures and variables needed to be accessed by both the C and assembly-language routines must be initialized. This will ensure the

data of being placed in the _DATA segment instead of the c_common segment.

A small-model program was mentioned before to contain two segments, but there are seven segments defined in Fig. 13. This is possible because of "groups", which allows the combination of several segments into one. The _DATA, CONST, _BSS, c_common, NULL, and STACK segments are grouped together by the Microsoft C compiler into a group named **DGROUP.** The advantage of groups is that any data in a group can be accessed by the same segment register. Instead of needing two words to access a different piece of data, only the offset in the segment is needed. Thus, access time is nearly halved. After a data transfer of hundreds of words, the time saved can be significant. To summarize what segments are to be used in the assembly-language routines, it can be said that all code shall be in the _TEXT segment, and all global data items will be in the _DATA segment which is grouped into the DGROUP. Listing files of compiles and links will show segment and group names of all data.

It will be necessary to pass arguments from C to assembly-language routines. Arguments are passed on the stack, and the called routine must pop the arguments off the stack. Since words are only pushed and popped from the stack, the number of words pushed for the different data types is needed. Fig. 14 gives the number of words used for the standard data types in C.

data type	Number of words pushed on stack
char, short, int, signed char, signed short, signed int, unsigned char, unsigned short, unsigned int	1 word
long, unsigned long	2 words
float, double	4 words

Figure 14. Argument lengths of C data types.

When C calls an assembly-language routine, the last argument is pushed first and the first argument last. If an argument requires more than one word, the high word is pushed first, followed by the next higher word.

When a C program passes control to an assembly-language routine, certain registers must be preserved or changed before arguments can be grabbed and the routine executed. The BP, SI, and DI registers need to be saved, and the BP register should be set to the current SP register. Any segment values changed in the routine, such as SS, DS, or CS, should also be saved and then restored on exit from the routine. After the entry sequence, the arguments passed can be found on the stack starting at offset bp+4. Each argument must be popped off the stack according to the guidelines of Fig. 14.

With the arguments off the stack, the assembly-language routine then goes about its business. There is one extremely important convention when C and assembly need to use a common global variable or function. The global variable names and function names in the assembly language routine must be prepended by an underscore (_). If an assembly-language routine is required to access a global data variable named RESULT declared in a C program, the assembly routine must declare _RESULT as external in the _DATA segment. The routine can then modify the variable. The same applies toward routine names. If the C program wants to execute an assembly-language routine, the routine name must begin with an underscore. The C function call DO_IT() will access the assembly language routine named _DO_IT, as long as DO_IT() has been declared external in the calling C program. To summarize, if the assembly-language name does not begin with an underscore, it cannot be accessed in a C program.

As in the calling of C functions from other C programs, arguments are not automatically passed from assembly-language routines when returning to a calling C program. However, a value may be returned to a calling C program by placing the return value in the AX register. This works when returning characters, integers, or shorts, whether signed or unsigned. If a long, float, double, or pointer needs to be returned, see [4] for further information.

After the return value has been put in ax, the assembly-language must restore registers that were preserved on entry to the routine. Si, di, sp, and bp are restored before the return command is given.

An example of an assembly-language/C interface is shown in Fig. 15.

```
/**** C program source file ****/

extern int RESULT;
extern int DO_IT();

main()
{
   int error, x, y;

   x = 3;
   y = 5;

   error = DO_IT(x,y);

   if (error != 0)
      printf("an error has occurred\n");

   else
      printf("the answer is %d\n",RESULT);
}


;**** assembly language source file ****

_DATA SEGMENT WORD PUBLIC 'DATA'
extrn _RESULT:word
_DATA ENDS

_TEXT SEGMENT BYTE PUBLIC 'CODE'

PUBLIC _DO_IT
_DO_IT proc near

        push bp                              ; entry from C.
        mov  bp,sp
        sub  sp,8
        push di
        push si

        mov  ax,word ptr [bp+4]              ; pop x
        mov  bx,word ptr [bp+6]              ; pop y

        (operate on x and y in ax and bx, and put result in bx)

        mov  ds:[_RESULT],bx                 ; put answer in RESULT.
        mov  ax,0                            ; return no error.

        pop  si                              ; exit to C.
        pop  di
        pop  ds
        pop  bp

        ret                                  ; return

_DO_IT endp

_TEXT ends
```

Figure 15. Example C-assembly interface.

3.4 Interrupting the DACI, and data transfer.

The first three sections of this chapter have been devoted to explaining what the communication routines need to accomplish, and the tools available to help accomplish the tasks. In this section, the routines to implement the four commands discussed in Sec. 2.2 are developed.

Five routines have been written to accomplish relatively low-level tasks. These routines handle operations such as inputting and outputting bytes, inshaking and outshaking, and sending interrupt bytes. The instruction sequences given in Sec. 3.2 are used heavily in these five routines. Once the five "tools" were developed, the four command driver routines could be designed. These routines are the actual ones called from the main C program when one of the four commands is desired. The four command driver routines and the five low level routines will be included in the same source file for convenience and ease of maintenance. The name of the source file is COM_EHSI.ASM, which stands for: "Communication Routines for the EHSI Development System." Before the code is entered in the source file, some important commands need to be issued to conform with the interfacing restrictions described in the previous section.

COM_EHSI.ASM header

Before an 8088 op code is written, the segment and

group names need to be set to comply with the rules of interfacing C and assembly.

The _TEXT and _DATA segments are assigned the same align type, combine class, class name, and group so that code and data put in the segments are combined with the corresponding C segments at link time. Two variables from the C program are declared to be external words in the _DATA segment:

_SCREEN is the base address of the SCREEN array, which is the array of words sent via interface Command 2.

_data_pkg is the base address of the DATA_PKG structure, which is the destination for data received through interface command #1.

The CONST, _BSS, NULL, and _DATA segments are combined into a group named DGROUP, just as the C programs are grouped. Also, the code segment is assumed to be the _TEXT segment. The data, stack, and extra segment are all assumed to be the value of DGROUP.

The compiler directive **EQU** is used to make the assembly code more readable and easier to maintain. Any constants can be EQUated to an expression. On the first pass of the compiler, the expression is replaced by the constant equated to it earlier. The port addresses, enable and disable values, and error definitions are given expressions, hopefully descriptive of their functions.

The last line of the header opens up the _TEXT segment

so that all the following code is placed in the _TEXT segment. The line following the last routine in the source file tells the compiler to stop putting code in the _TEXT segment.

<u>COM_EHSI.ASM low level routines</u>

inshake_proc is a procedure that watches for the DACI to pulse the Z-158 INSHAKE line. Fig. 16 shows the flow of the procedure. Since the line is normally high, the routine waits for the line to go low. If this doesn't occur within a certain amount of time, an error is returned in ax. The time that is allowed for the line to go from high to low is controlled by an expression called INSHAKE_WAIT_LOW. The value of the expression is defined in the header of COM_EHSI.ASM, and the equation to determine the actual time is found in Appendix B. After the INSHAKE line goes low, the routine watches for the transition back to high. An error is returned by ax to the calling routine if the line does not go back high within a specified amount of time. INSHAKE_WAIT_HIGH controls the length of time to wait for the line to go back high. See Appendix B for more information. If no errors occurred, NO_ERROR is returned in ax to the calling routine. This procedure is only called by other assembly language routines. Therefore, special entry and exit procedures for this routine are not necessary. A 'return' instruction transfers control back to the calling routine.

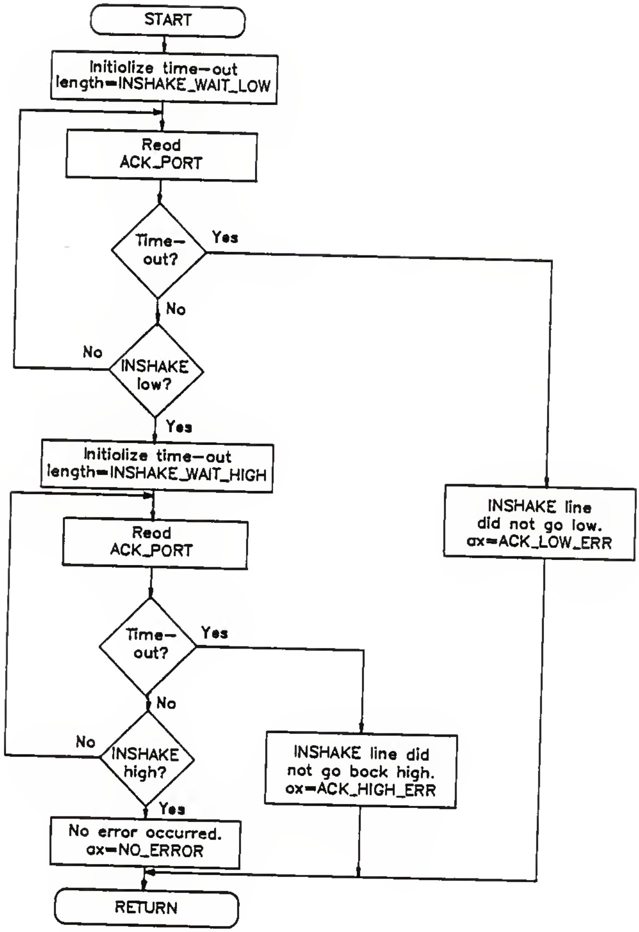

Figure 16. inshake_proc flowchart.

outshake_proc is a procedure that outputs a pulse on the OUTSHAKE line of the Z-158. Fig. 17 shows a flowchart of the procedure. The line is normally high, so the pulse consists of a transition from high to low, and then back high. The time that the line is low is controlled by an expression defined in the header. Errors cannot occur, so nothing is returned. This procedure is only called by other assembly language routines. Therefore, special entry and exit procedures for this routine are not necessary. A 'return' instruction transfers control back to the calling routine.

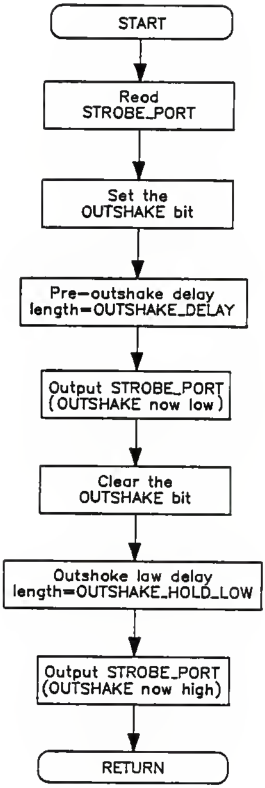

Figure 17. outshake_proc flowchart.

input_byte_proc is a procedure that performs the steps necessary to read the data bus. Fig. 18 is a flowchart of the procedure. The data bus is enabled for a read, the read is performed, and the data bus is disabled. outshake_proc is then called to tell the DACI that the read was accomplished. No errors can occur, so an error code is not returned. The byte read from the data bus is returned in ax. This procedure is only called by other assembly language routines. Therefore, special entry and exit procedures for this routine are not necessary. A 'return' instruction transfers control back to the calling routine.

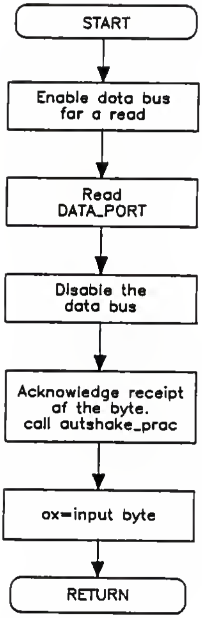

Figure 18. input_byte_proc flowchart.

output_byte_proc is a procedure that outputs a byte to the data bus and then waits for the acknowledge pulse from the DACI on the Z-158 INSHAKE line. Fig. 19 is a flowchart of the procedure. The byte to output is passed to the routine in al. The data bus is enabled for a write, and the output byte is put on the bus. outshake_proc is called to let the DACI know that a byte is on the data bus waiting to be read. Before disabling the data bus, the routine must wait for the DACI to acknowledge that it read the byte. The procedure inshake_proc is called to wait for the acknowledge pulse on the INSHAKE line. If an error occurred in receiving the acknowledge pulse, the error returned from inshake_proc is kept in ax to be passed to the calling routine. The data bus is disabled after inshake_proc. This procedure is only called by other assembly language routines, so the special entry and exit sequences required for calls from C programs are not necessary. A 'return' instruction transfers control back to the calling routine.

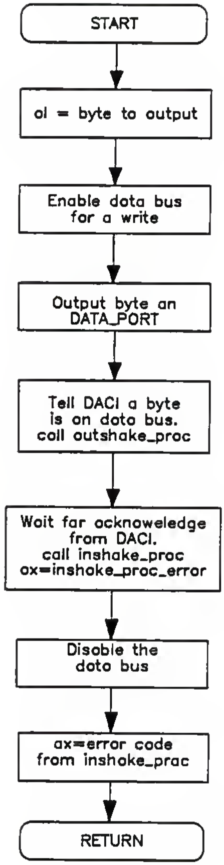

Figure 19. output_byte_proc flowchart.

output_int_byte_proc is a procedure used for interrupting the DACI to initiate one of the four interface command routines. Fig. 20 is a flowchart of the procedure. After enabling the data bus, the interrupt vector passed by the calling routine in al is placed on the data bus. To interrupt the DACI, the IRQOUT line is pulsed low and back high. The routine must then wait for an acknowledge pulse from the DACI. From the timing diagrams in Sec. 2.2, the DACI can take up to 135 usec to respond to the interrupt pulse. If an acknowledge error occurs, the routine returns an error code to the calling function. If the acknowledge pulse was received without error, a bus check is performed to make sure the DACI is thinking the same thing the Z-158 is. The data bus is cleared by writing 00H to it, and then a read is performed to make sure the DACI is not trying to output something. If 00H is read in, then no error occurred. If a non-zero byte was read, a bus conflict error is returned to the calling routine. All error checking is completed, and the data bus is disabled. This routine is called only by other assembly language routines, so special entry and exit sequences are not necessary. A 'return' instruction transfers control back to the calling routine.

The five procedures have been coded and can be found in COM_EHSI.ASM, Appendix A. The sequences written in Sec. 3.2 are used throughout the routines. Also, the expressions

used for port addresses, output control bytes, and error codes are defined in the header portion of COM_EHSI.ASM.

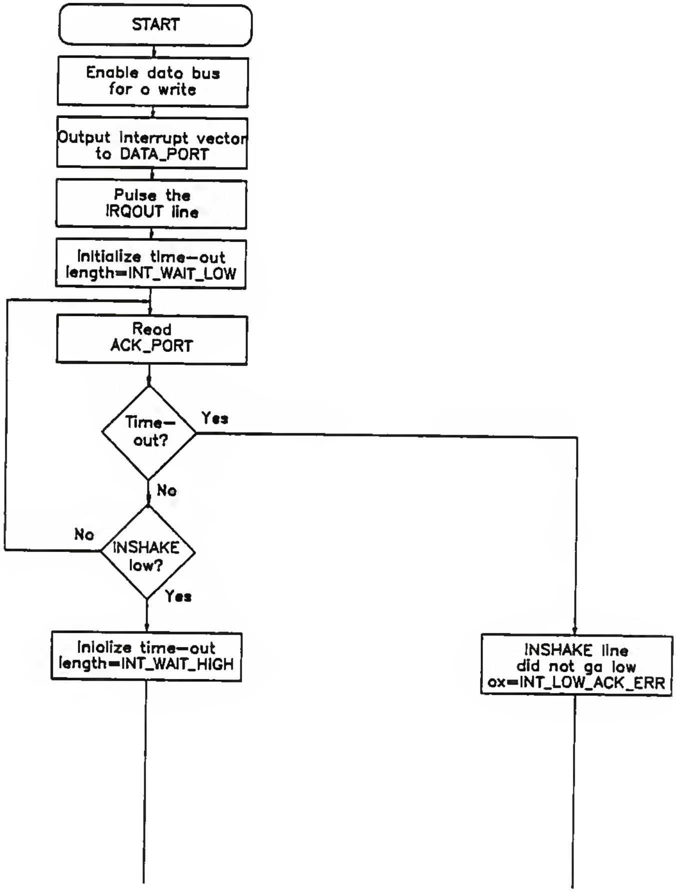

Figure 20. output_int_byte_proc flowchart

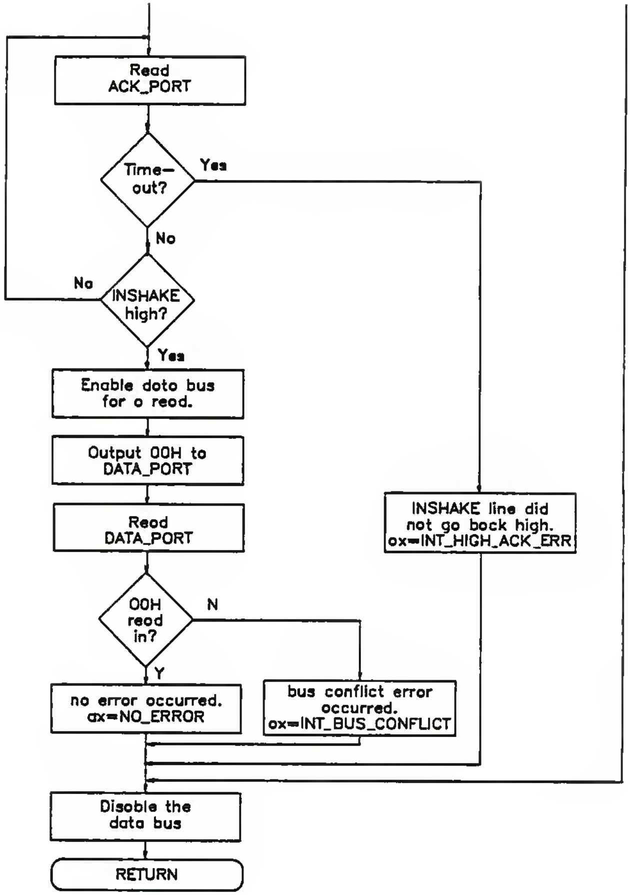

Figure 20. output_int_byte_proc flowchart (cont.)

COM_EHSI.ASM Command routines

The development of assembly-language routines to implement the four commands discussed earlier can now be described. The previous three sections have laid the groundwork for the commands. The timing diagrams (Fig. 4, 5, 6, 7) will be adhered to.

GET_DATA_PACKAGE is the procedure called from a C program to execute command #1. The data package is to be transferred from the DACI to the Z-158. A structure named data_pkg is the final destination of the transferred data. Fig. 21 gives a flowchart of the procedure. Since these procedures are called from C programs, the entry sequence explained in Sec. 3.3 is used. There are no arguments passed to this routine. First, the starting address of the data_pkg structure is retrieved. output_int_byte_proc is then called to output the interrupt vector 01H to the DACI. If the interrupt was unsuccessful, an error value is returned to the C program via the ax register. The routine will jump to the exit sequence after the error. If the interrupt was successful, the DACI will then start the data transfer. inshake_proc is called to wait for the pulse signalling the number of bytes to be sent is on the data bus. If an error is returned from inshake_proc, the error handling procedure is the same as with output_int_byte_proc. The number of entries is checked with the known value, and an error is returned if there is a discrepancy. The routine

reads the data after each outshake pulse is detected, until all the data has been received. Acknowledge pulses are sent on the OUTSHAKE line after each byte is received. After the transfer is complete, the routine must execute the exit sequence described in Section 3.3. This will restore the registers saved in the entry sequence. A 'return' instruction transfers control back to the calling C program.

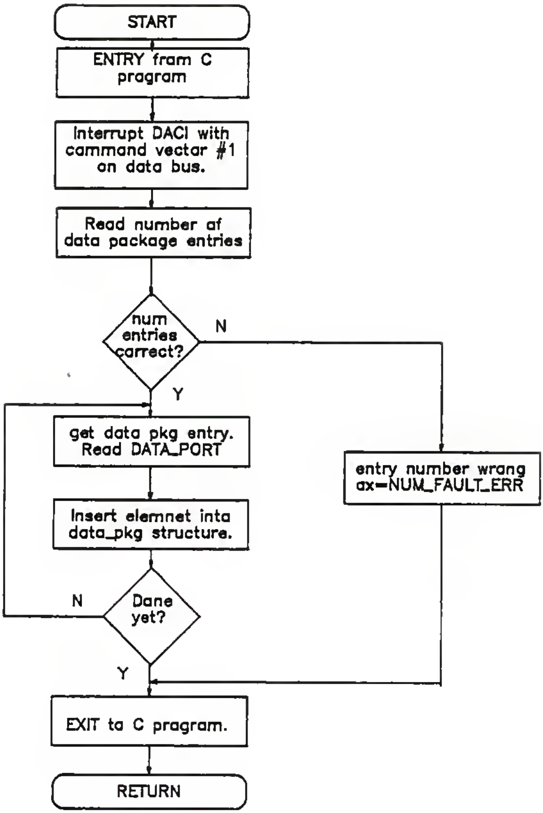

Figure 21. GET_DATA_PACKAGE flowchart.

SEND_SCREEN is the procedure called from a C program to execute Command 2. The contents of the SCREEN array are to be transferred from the Z-158 to the DACI. Fig. 22 is a flowchart of the routine. Data words are sent to the DACI until the MSB is OFFH. This is the End of SCREEN signal. As with Command 1, the entry sequence from C programs must be executed since this routine will be called from C programs. After the entry sequence, output_int_byte_proc is called to output the interrupt vector corresponding to Command 2. If the interrupt was successful, the data words in SCREEN are sent by bytes, first the MSB and then the LSB. If an error occurs either in output_int_byte_proc or output_byte_proc, the transfer is suspended and an error code is returned. In any case, before the routine is finished, the exit sequence must be executed to restore the C register to their original values. A 'return' statement then transfers control back to the calling C program.

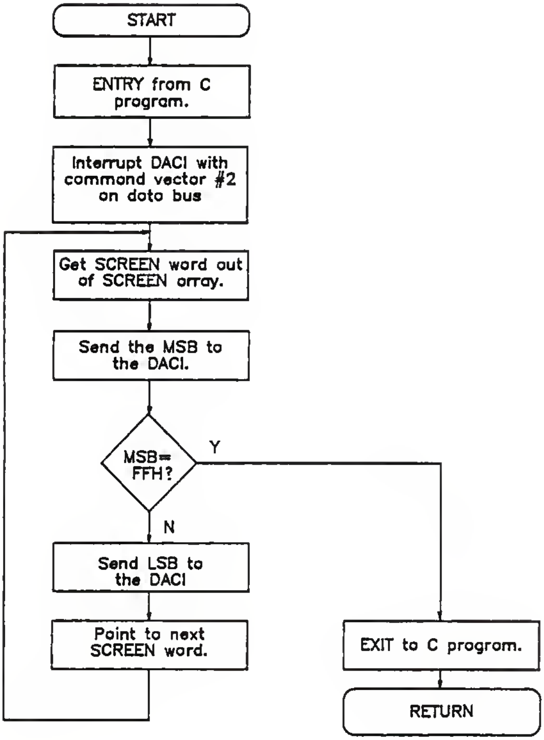

Figure 22. SEND_SCREEN flowchart.

RETRIEVE_SCREEN is the procedure called from a C program to retrieve display code that is currently in HP-1345A VGM. Fig. 23 is a flowchart of the procedure. After the entry sequence, which is required for routines being called from C routines, output_int_byte_proc is called to output the interrupt vector corresponding to Command 3. The beginning address of the code to retrieve is found in SCREEN[0], and the end address is in SCREEN[1]. The transfer is then conducted by bytes, first the MSB and then LSB. The retrieved screen code is placed in the SCREEN array starting with the first element. If an error occurs, an error code is returned in ax. If no errors occur, the exit sequence to restore the C registers is executed. A 'return' instruction transfers control back to the calling C program.

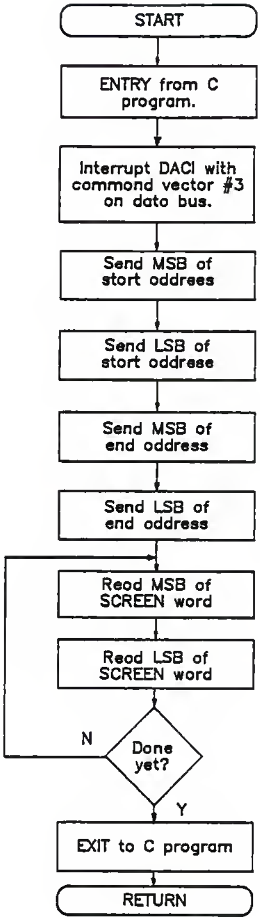

Figure 23. RETRIEVE_SCREEN flowchart.

TOGGLE_ALARM_SWITCH is the procedure called from a C program to toggle the ON/OFF status of the alarm. Fig. 24 is a flowchart of the procedure. After the entry sequence is completed, the Z-158 only needs to have output_int_byte_proc output the interrupt vector corresponding to Command 4. This will cause the DACI to toggle the alarm state. The exit sequence is executed to restore the C registers, and then a 'return' instruction transfers control back to the C program. If an error was returned from output_int_byte_proc, the error code is returned to the calling program.

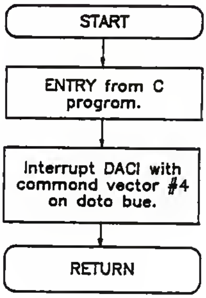

Figure 24. TOGGLE_ALARM_SWITCH flowchart.

The four routines have been implemented in code and can be found in COM_EHSI.ASM, Appendix A, along with the low level routines. Error codes and other expressions can be found in the header at the beginning of the source file.

3.5 Servicing Interrupts from the DACI

Several routines will be presented in this section that deal with the setup, execution of, and restoration of the interrupt environment needed for the EHSI development system. The DACI interrupts the Z-158 by pulsing its IRQOUT line with an interrupt vector on the data bus (see Fig. 2). The Z-158 must suspend its current process and service the interrupt. The routine that does the servicing is installed at startup time into the DOS environment as interrupt OFH, the parallel port interrupt. After the servicing is complete, the suspended process is continued without knowing that the interrupt occurred. It will find out though, when it checks a special stack which holds received interrupt vectors. After a shutdown vector is received, the interrupt environment is returned to its original state. INT_EHSI.ASM is the assembly-language source file that contains the procedures discussed in this section. Header information will be discussed first, followed by the presentation of each routine.

INT_EHSI.ASM header

At the top of INT_EHSI.ASM is a header similar to the one found in COM_EHSI.ASM. The segment and group names necessary to allow C and assembly to communicate are assigned according to the guidelines set forth in Sec. 3.3. Since the implementation of the correct segment and group names were done in the COM_EHSI.ASM header, see its explanation in the preceding section for details. There are some different variables in the header that do need explanation. Args is an expression that holds the offset of the first argument from the base of the stack after entry into an assembly language routine called from a C program. The value is for a small-model program.

Three double-word locations are initialized to 0 and inserted into the _DATA segment. These locations are for addresses that are saved as static variables, so that the three routines discussed shortly may have access to the same addresses. The values cannot be disturbed by any other routines. The function of each is described below:

int_OF — is where the address of the old interrupt OFH handling routine is stored during operation of the EHSI Development System. At shut-down this address is reinstalled into the interrupt address table.

int_depth is the address of the variable in the C program that keeps track of the interrupt vector stack depth.

int_stack is the address of the first element of the interrupt vector stack created in the main C program.

INT EHSI.ASM interrupt routines

INITIALIZE is a routine called from the main C program to set up the Z-158 so that interrupts received on the IRQIN line can be serviced. The setup includes installing HANDLER as the servicing routine. Fig. 25 is a flowchart of the routine. Since this routine is called by a C program, the entry sequence covered in Sec. 3.3 must be executed first. Two arguments are being passed by the calling C program. The last argument, the address of the interrupt vector stack, is popped off the stack first. The address of int_depth is popped last since it was the first argument in the call.

Before installing the new interrupt OFH service routine, the old routine's address is fetched from the interrupt vector table and saved for its reinstallment after system shut-down. The new interrupt OF servicing routine, named HANDLER, is now installed by placing HANDLER's address in the interrupt vector table in low memory.

Interrupt OF is channelled through an 8259A

Programmable Interrupt Controller (PIC). The PIC receives and prioritizes interrupt numbers 08 through 0F. When the PIC detects an interrupt on one of its lines, if there are no other interrupts pending, the PIC will interrupt the 8088 and pass on the interrupt number. If two interrupts hit the PIC at the same time, the one with the higher priority is serviced first.

The PIC contains an Interrupt Mask Register (IMR) that needs to be changed. The parallel port interrupt must be enabled, and the system clock needs to be ignored during operation of the EHSI Development System. The internal clock interrupts the system over 18 times per second for updating purposes, and the update routine causes timing problems for the Z-158 -to- DACI routines. Therefore, the clock is temporarily disabled via the IMR. The procedure for changing the IMR is explained in Sec. 3.2.

The output control port (Port 037AH) needs to be initialized so that control lines are in their default levels. Line D4 of Port 037AH is the IRQ_ENABLE line. Sec. 3.2 discusses what it needs to be to enable interrupts on the parallel port interrupt line. Initialization is now complete. The exit sequence necessary for restoring C registers is executed, and a 'return' instruction transfers control back to the calling C program.

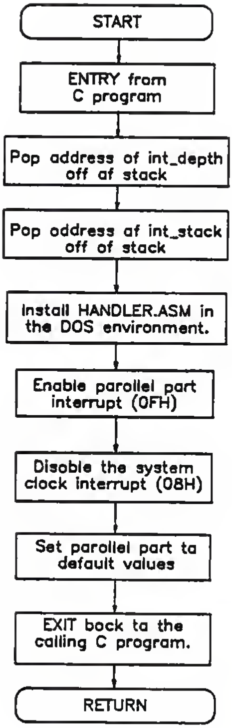

Figure 25. INITIALIZE flowchart.

HANDLER is the interrupt service routine installed by INITIALIZE to service interrupts from the DACI. This routine is not executed directly from any C or assembly language routine. The procedure for servicing the interrupt is shown in the flowchart of Fig. 26. All interrupts are turned off immediately in the routine. This will ensure that the interrupt service routine is not interrupted. All registers used by HANDLER must be saved before the actual servicing, so that when the suspended process is continued, registers will not be mysteriously changed. The data bus is then read to retrieve the interrupt vector from the DACI. The OUTSHAKE line is taken low to acknowledge receipt of the vector.

The interrupt vector needs to be placed appropriately in the interrupt vector stack. The current depth of the interrupt vector stack is retrieved and incremented to get the depth of the current vector. Since the vectors are stored as words (2 bytes), the offset of the current vector on the stack will be twice the current depth. After the offset is calculated, the base address of the interrupt vector stack is fetched. The offset is added to the stack base address, and this is the destination for the received interrupt vector.

There are two conditions that affect placement of the received interrupt vector in the interrupt vector stack. If the interrupt vector stack is full, then the current

interrupt is ignored by not placing it in the stack. If the shut-down vector is received, priority is given to it by placing it at the bottom of the stack so that it will be next in line to be serviced.

Since the DACI needs an acknowledge pulse of at least 23 microseconds, a short delay is performed before bringing OUTSHAKE back high. One last duty must be performed before executing the exit sequence. An End of Interrupt signal is sent to the 8259A PIC so it can resume operation. The process, written out in Sec. 3.2, is like an "outshake" pulse to the PIC to acknowledge the interrupt. The registers saved during entry are then restored, and interrupts are turned back on. The 'iret' instruction transfers control back to the suspended process in the exact state it was interrupted in.

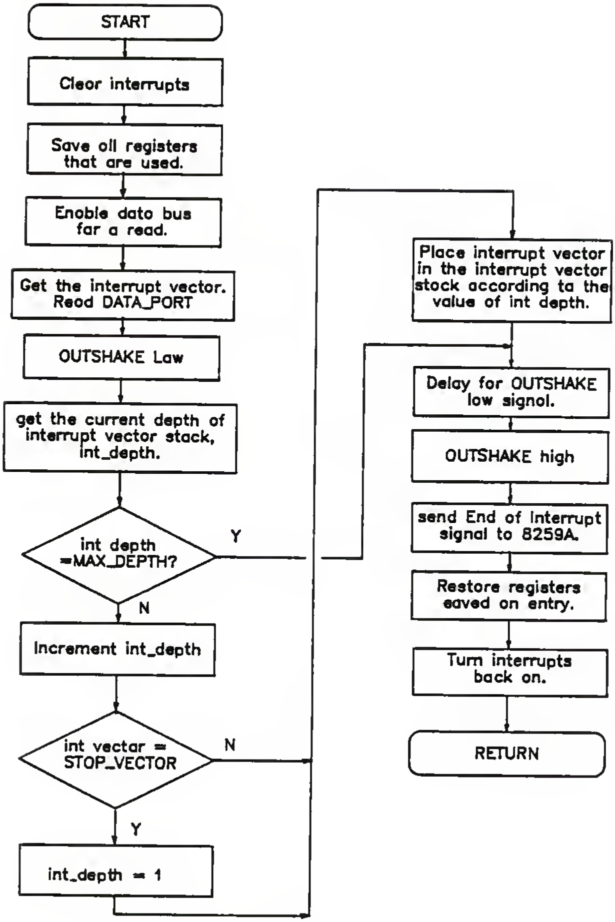

Figure 26. HANDLER flowchart.

RESTORE is the routine called to restore the operating environment back to the state before INITIALIZE was called. The routine is called after the system shut-down vector is received. Since this routine is called from a C program, the entry sequence for calls from C must be executed first. The old interrupt OF address, saved in INITIALIZE as int_OF, is reinstalled in the interrupt vector table in low memory. The PIC Interrupt Mask Register is also restored to its original state. The exit sequence to restore the C program registers is executed, and a 'return' instruction transfers control back to the calling program.

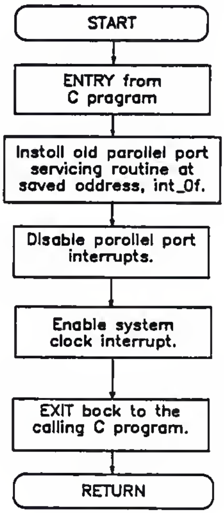

Figure 27. RESTORE flowchart.

IV. THE MAIN PROGRAM

4.1 Purpose of EHSI.C

The bulk of this report has been devoted to communication routines, but these would be useless if there were not routines to call upon them. Ehsi.c is the main program that controls the EHSI development system. The main program calls other functions to service the interrupts from the DACI. The routines that are called from ehsi.c use the four commands discussed earlier to talk with the DACI. There are three different classes of functions that are called on receipt of an interrupt vector.

If the start-up or shut-down vector is received, initialization or restoration is performed respectively. If the system clock interrupt is received, a function is called to update the current page. This usually involves a call to GET_DATA_PACKAGE, HP-1345A code generation, and then a call to SEND_SCREEN. If a key number is received, a specific function to service that key is invoked. The key routines will be discussed in Chapter 5. The flow chart of ehsi.c is shown in Fig. 28. The following sections shall explain what goes on in each block of the program.

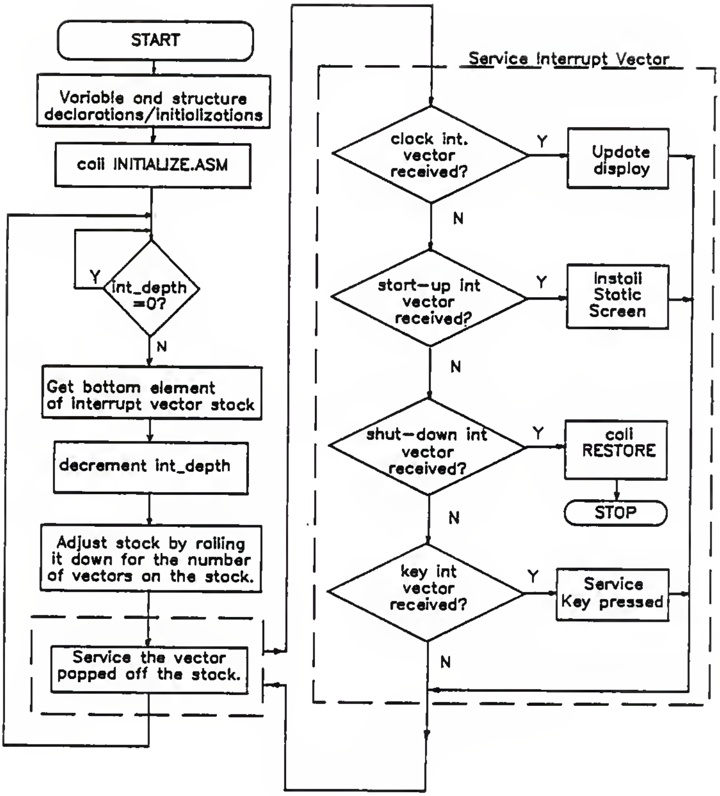

Figure 28. Ehsi.c flowchart.

4.2 Declarations and initializations.

The main program is responsible for declaring all the global data used in the system, and it is logical to declare other variables and flags along with them at the beginning of the program. "data_str.h" is the include file that declares the structures needed. data_pkg is a structure of type DATA_PKG which holds the current data package when interface Command 1, GET_DATA_PACKAGE, is called. The other two structures defined were added by C. Robertson [3] in his work. The flags defined at the bottom are also his. Notice that data_pkg has been initialized to zero. This must be done, or else the structure will end up in the c_common segment where the assembly can't access it. SCREEN[] is initialized to OFFFFH and a size of 1000 words. It is initialized also to insure placement in the _DATA segment.

After the declarations, but before polling the interrupt vector stack can occur, INITIALIZE is called to set up the environment and install the interrupt service routine, HANDLER. The system is now ready to service interrupts from the DACI.

4.3 Ehsi.c Interrupt Servicing.

Ehsi.c polls the variable int_depth, waiting for it to become greater than zero. A greater than zero value tells the main program that an interrupt vector is present on the stack. The vector is popped and placed in a variable named

int_number. int_depth is decremented to tell HANDLER where to put the next vector. If int_depth is still greater than zero after the decrement, the stack is rolled down to put the next interrupt vector in the queue.

The interrupt vector stack operations are completed for the current vector. The rest of ehsi.c is devoted to determining what course of action to take with the current vector. If the startup vector is received, the static parts of the displays are installed in display memory. If the shutdown vector is received, RESTORE is called to restore the DOS environment back to normal. The program will also terminate. On receipt of the clock interrupt vector, Ehsi.c calls a function to perform an update of the currently displayed page. dat_pg_dynamic, nav_pg_dynamic, and ils_pg_dynamic are three functions written by Robertson [3], that update the dynamic parts of the displays. The final type of interrupt that can be received is the keyboard interrupt. Another decision statement is used to determine which key was pressed and what kind of action is needed. The "service key" block in the flow chart of Fig. 28 will be expanded in the next chapter.

After the interrupt vector has been serviced, the main program returns to polling int_depth, waiting for the signal that another interrupt vector is on the interrupt vector stack.

4.4 Ehsi.c considerations.

The claim was made earlier that the system is running in an interrupt environment, but it can be seen from the main program, ehsi.c, that some polling does occur. This hardly hampers system performance because all that is polled is one memory location. int_depth can be checked every couple microseconds when the interrupt vector stack is empty. When vectors do appear on the stack, they are executed as quickly as the service functions can be completed. In the case where there is more than one vector on the stack, the bottom vector will be serviced first. As soon as that service is completed, the program will return to see that there is still a vector on the stack. The second vector will then be executed immediately.

The program continues until the shut-down vector is received from the DACI. As mentioned before, RESTORE is called to return the system to its original state. Ehsi.c then terminates.

V. KEY SERVICING ROUTINES

5.1 Purpose of key routines

An important part of the EHSI Development System is the command keyboard. Figure 29 shows how the 6x6 keyboard is set up. Several of the keys deal with changing displays, and some enter flight-related data. Others operate a clock timer implemented by Robertson [3]. A Hewlett Packard style calculator has been implemented by the author as a means of keeping data entries in a stack environment.

Two variables, x_buffer and y_buffer, are declared in ehsi.c as the elements of a small "calculator" stack to be shared by all the key servicing routines. x-buffer will be referred to as the first, or bottom element. Key_buffer is a character string that keeps track of the current numbers being displayed on the "command line" of the data page. Chapter 7 will show the data page display with the command line in use.

5.2 HP-1345A memory organization

In the key routines it is necessary to generate screen code for the display. Two routines, string_gen() and insert(), were written by Robertson [3] in his work. These routines generate HP-1345A code for character strings, and the code words are placed in the SCREEN[] array, but the calling routine is responsible for the address pointer in the display memory. The organization of the HP-1345A

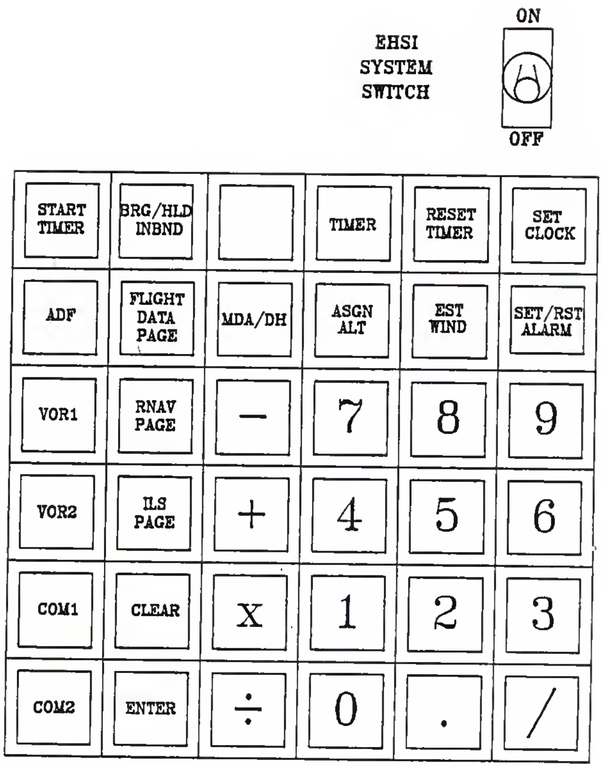

Figure 29. 36 Key command keyboard.

"Vector Graphics Memory" for the EHSI development system is presented below.

A memory map of the VGM is shown in Fig. 30. For the EHSI development system, all the static display information is stored in static memory during system operation. Dynamic display information is stored in locations that are often changed. Jump vectors are placed throughout memory to tell the HP-1345A which commands are to be displayed on the screen. At address 000H, a jump vector tells the HP-1345A to display the static parts of one of the display pages. The static sections are installed upon receipt of the start-up interrupt vector from the DACI. After the static display commands are performed, a jump vector transfers the HP-1345A pointer to the dynamic part of the currently displayed page. After these commands are completed, a jump vector points to the command line memory. After the command line is displayed, a jump to the end (FFFH) is performed. The only rule with jump vectors is that a jump vector cannot be jumped to. Therefore, location FFFH contains a NO-OP. The next location (000H) starts the refresh operation again by repeating the process of jumping to static display memory.

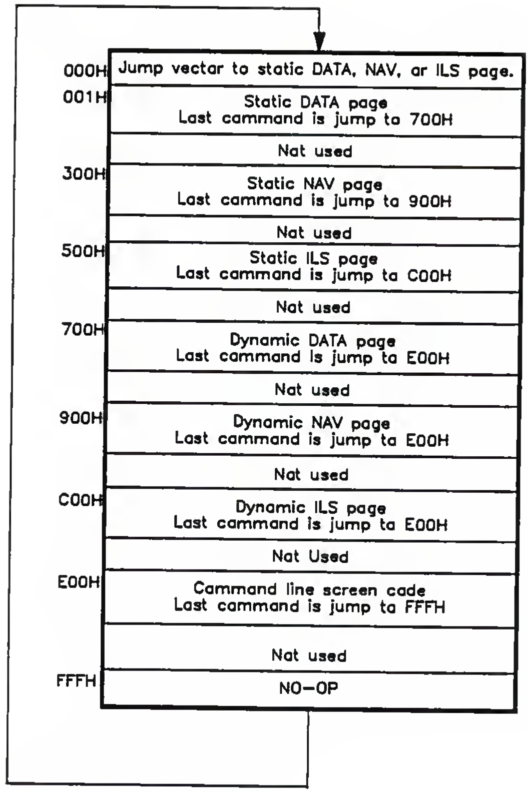

Figure 30. HP-1345A memory map.

5.3 Description of key routines

The key servicing routines that the author has implemented are discussed below.

update_key_buffer is a function called when one of the digits or the decimal point key is hit. The routine determines which key was hit, and then adds the character to the end of key_buffer. The updated key_buffer is then converted to screen code and sent to the display via SEND_SCREEN.

roll_stack is a routine called when the ENTR key is hit. This function moves the old value of x_buffer into y_buffer, and the number currently displayed is put into x_buffer. Remember that the number being displayed is actually stored as a character string in key_buffer. Therefore, a conversion function is called to convert the character string in key_buffer to a number. key_buffer is then cleared, and SEND_SCREEN is called to clear the command line.

clear_stack is a routine called when the CLEAR key is hit. This function clears the contents of x_buffer, y_buffer, and key_buffer. SEND_SCREEN is called to clear the command line.

do_math is a routine called when the addition, subtraction, multiplication or division key is hit. The specified math function is performed on the contents of y_buffer and x_buffer, and the result is placed into

x_buffer. The result is also placed on the command line via SEND_SCREEN.

display_data_page, **display_nav_page**, and **display_ils_page** are routines that use SEND_SCREEN to change jump vectors in vector memory so that the respective pages are displayed.

reset_alarm is a routine that is called when the RESET ALARM key is hit. TOGGLE_ALARM_SWITCH is invoked to toggle the ON/OFF state of the alarm on the interface board.

call_cmd3 is a routine written for the purpose of testing interface command 3, RETRIEVE_SCREEN. Since the routine is not yet being used in the main part of the system this routine shows the operation of command 3. The start address of the retrieve is taken from y_buffer, and the end address is in x_buffer. The call to RETRIEVE_SCREEN is performed, and the retrieved code is printed on the Z-158 screen.

Fig. 31. shows a flowchart of the decision making being done in the "key service" block of Fig. 28. Routines not explained above have been written by Robertson [3].

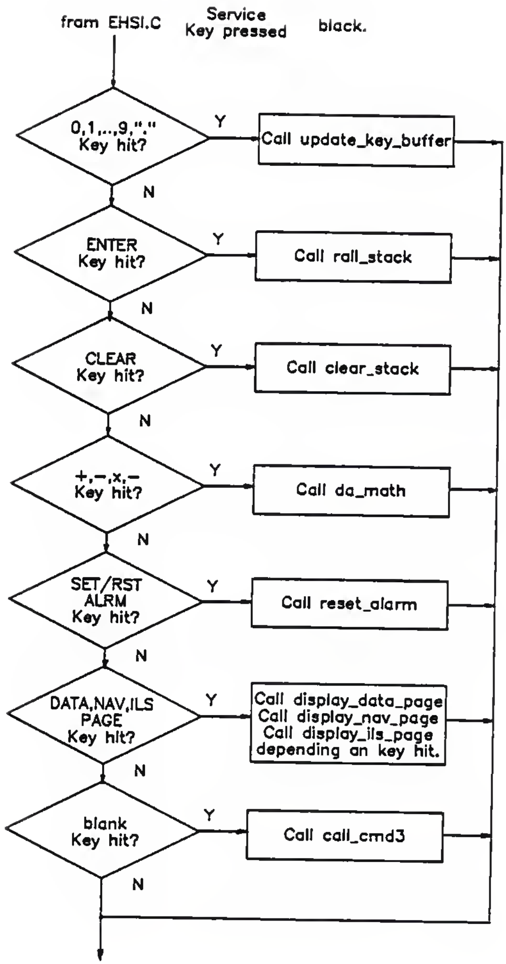

Figure 31. Key interrupt servicing flowchart.

VI. SOFTWARE CONDITIONING OF INPUT SIGNALS

6.1 Flight Data Sampling and Filtering

Filtering of analog flight data has become of interest in the development of the EHSI system. Analog meters tend to have non-linearities and electrical zero-crossing points that are undesirable. A digital filter can be implemented in software to alleviate some of the problems. A filter could also be used to extract information from a signal that would be helpful to a pilot. Therefore, a routine has been written to sample and filter the flight data signals coming from the flight simulator in order to explore the many possibilities.

Flt_ehsi.c is a routine that allows a user to sample and filter a selected element of the DATA_PKG structure. The user must enter the desired filter order and coefficient values in an include file named flt_ehsi.h. Ehsi_filter is the function called by flt_ehsi to implement the filter. The executable file, flt_ehsi.exe, is created by issuing the command MAKE FLT_EHSI. After the executable file is created, the program is run with the flight simulator operational.

The user will be prompted to enter a number corresponding to the data element he wishes to sample. He will then enter the number of samples to take. The program will prompt him to enter the names of the files in which to

place the unfiltered and filtered data streams. After running the program, the user can upload the data files to the VAX and run a program named CONVERT.FOR to convert the data file from IBM-PC format to VAX-VMS format. RALPH2, a signal analysis program, can then be run to look at the data files. A number of functions can be invoked in RALPH2, one of them being PLOT. As an example, VERTICAL_SPEED was sampled at 2 Hz and filtered by a fourth-order moving average filter. The two data files created were uploaded to the VAX via KERMIT, converted to VAX format via CONVERT, and plotted by the FANCY PLOT function in RALPH2. The resultant plot is shown in Fig. 31. The lowpass nature of the filter is evident from the way some of the sharp peaks were smoothed out.

Using this method of sampling and filtering data signals from the flight simulator, some practical applications are looked at in the next two sections.

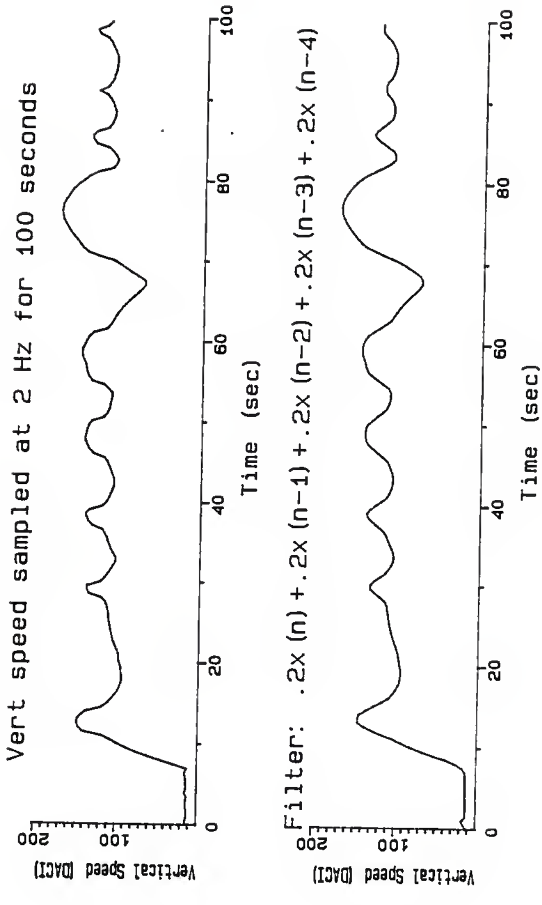

Figure 32. VERT_SPEED data sample.

6.2 A Low-pass filter for GLIDESLOPE

The GLIDESLOPE line coming out of the simulator was sampled at 20 Hz. for the purpose of extracting trend information. It was found, though, that a considerable amount of noise was present in the signal. The top graph of Fig. 33 shows the sampled data. A digital low-pass filter consisting of a ninth-order moving average filter was built to condition the signal. As can be seen by the bottom graph of Fig. 33, most of the noise has been removed, and the signal seems to have the same characteristics as the raw data.

This averaging process may be used on any of the data_pkg signals. The conditioned signals do not have the sudden changes that make displays "flicker", and the pilot can obtain more accurate flight data.

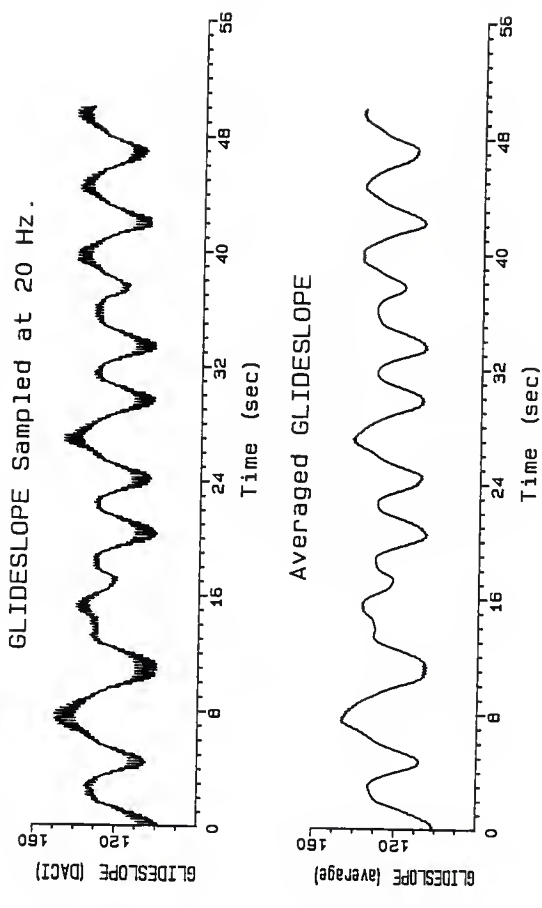

Figure 33. Low-pass filtered GLIDESLOPE.

6.3 A Differentiator for CDI and Glideslope

Two signals, CDI and GLIDESLOPE, are critical during the performance of an ILS approach. The trends of these two data streams could give the pilot information about the tendancies of the airplane to deviate from the approach path. Therefore, a digital filter differentiator was designed to extract this information.

A "low-noise" differentiator given in [5] has the following form:

$$3 \sum_{k=-N}^{k=N} k \frac{x(n-k)}{N(N+1)(2N+1)}$$

Setting N equal to 3, the coefficients were found to be:

$$y(n) = -.107x(n+3) - .0714x(n+2) - .0357x(n+1) \\ +.0357x(n-1) + .0714x(n-2) + .107x(n-3)$$

To force the filter to be causal, a time delay of three periods must be introduced. If a filter is operating in real-time, it cannot use input values that have not occurred yet. Such is the case with the x(n+_) components. This does introduce some error into the differentiation, since the "true" result is always 3 periods late. If sampling rates are sufficiently high, the delay is unnoticable in the cockpit. The digital differentiator used in the EHSI development system is written out below:

$$y(n) = -.107x(n) - .0714x(n-1) - .0357x(n-2) + .0357x(n-4) + .0714x(n-5) + .107x(n-6)$$

The CDI signal was sampled and filtered by the differentiator. The two plots are shown in Fig. 34. The diffentiator did a good job, but it magnified the small amount of noise present with the input signal. The output of the differentiator was then averaged and plotted in Fig. 35. The averaged signal gives a good indication of the trend of the CDI.

The averaged GLIDESLOPE signal found in Sec. 6.2 was also filtered by the differentiator. The results are shown in Fig. 36. The derivative of the averaged signal gives a good indication of the trend of the GLIDESLOPE. The approach taken here is different than the previous one used in differentiating CDI, but the two results are similar. As expected, though, the average of the differentiated signal (CDI) was smoother than the differentiated average (GLIDESLOPE).

With trend information available, an indicator can be placed on the ILS page to show the pilot his trend.

6.4 Filtering Considerations

It is worth noting some considerations concerning the filtering of flight signals. The EHSI Development System currently has a maximum refresh rate of 2 Hz., and a sampling rate this low makes real-time digital filtering

somewhat limited. Increasing speed by changing processors will be necessary to get the sampling rates higher so that useful information can be obtained. There will undoubtly be an abundance of noise present in the cockpit, so low-pass filtering and shielding will become important. As was shown in the plots, noise shows up quite readily when differentiating, so a differentiator with a cut-off may be needed to smooth the differentiated signals.

A simple way of displaying trend information with the differentiated CDI and GLIDESLOPE signals is to treat one as the real and the other the imaginary part of a complex number. The phase and magnitude of the complex number could be used to display an arrow with variable length that pointed according to the phase value. The length of the arrow could show how much of a change is occurring.

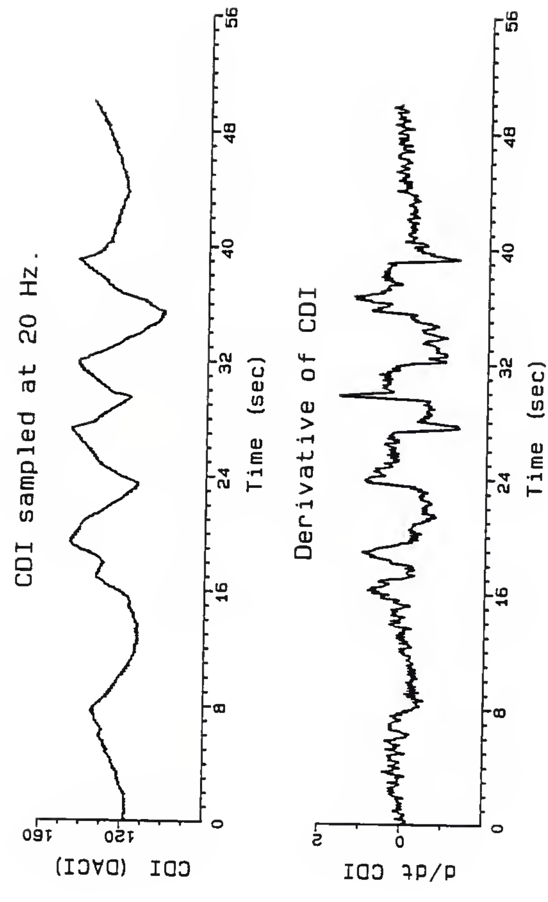

Figure. 34 Derivative of CDI.

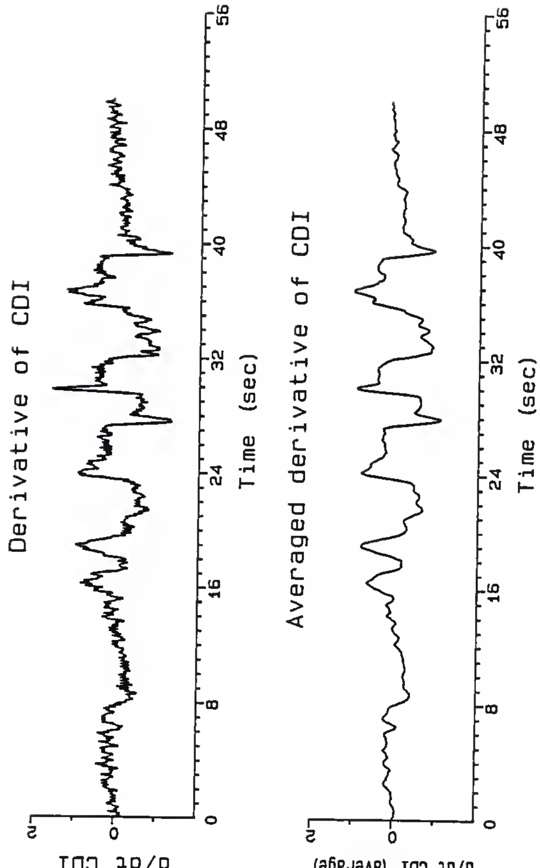

Figure 35. Averaged derivative of CDI.

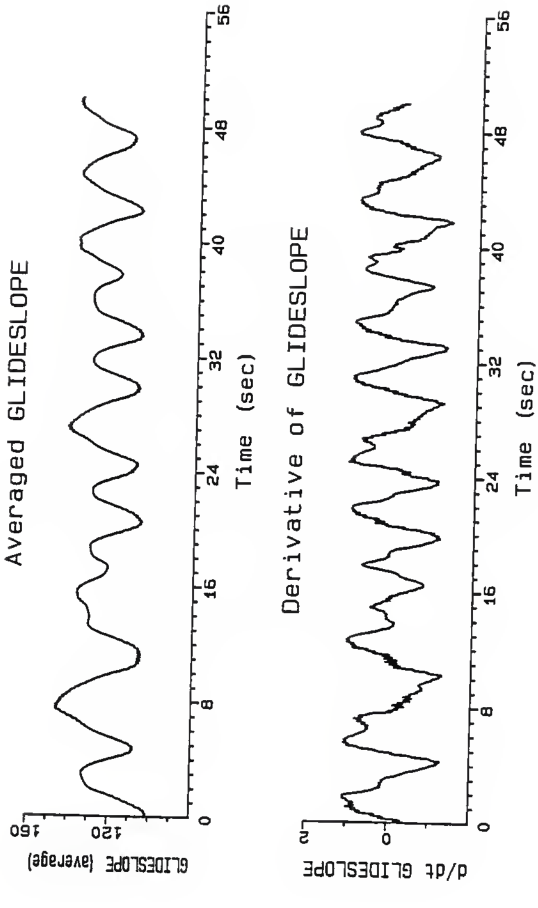

Figure 36. Derivative of averaged GLIDESLOPE.

VII. THE EHSI DISPLAY PAGES

Using the routines developed in this research, C. Robertson [3] has written fuctions in C to implement the different pages on the HP-1345A. He has also written routines to service key presses and operate alarms. The three display pages, Data page, NAV page, and ILS page, were proposed by Lagerberg [2]. Robertson has implemented variations of these concurrently with the research conducted in this thesis. Each page shall now be discussed briefly.

7.1 Data page

The Data page is designed to give general flight data and provide for a page to enter communication frequencies, timer values, and other keyboard entries available. Fig. 37 shows the proposed Data page [2]. Also available are engine statistics and weather information.

7.2 NAV Page

The Navigation page is designed to offer a view to the pilot of where he is relative to navigational fixes. A type of "road map" is displayed on the screen showing the plane relative to VOR's, NDB's, and preprogrammed waypoints. Fig. 38 shows the proposed NAV page [2]. The compass rotates as the plane changes course, and distance and direction information is constantly updated.

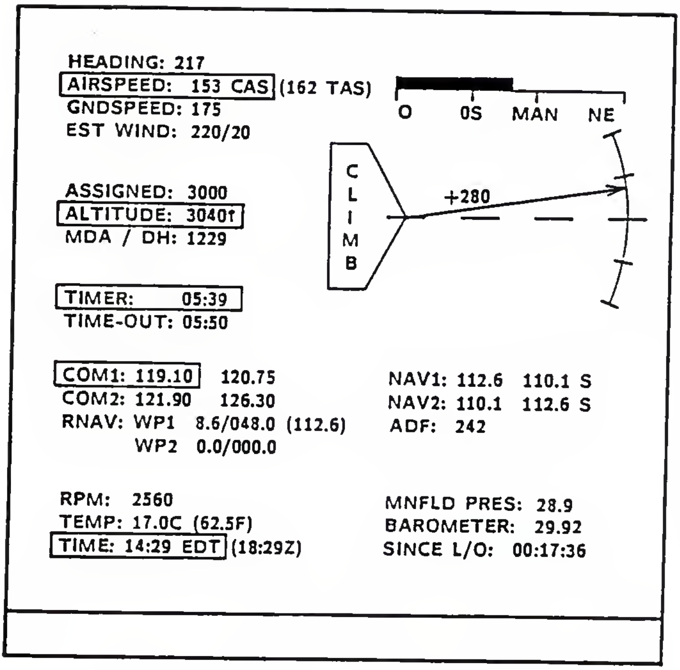

Figure 37. Proposed Data page.

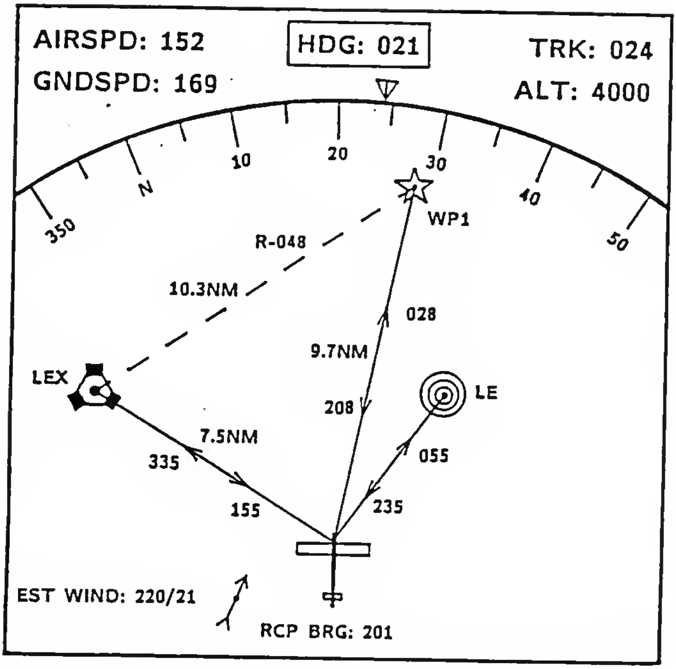

Figure 38. Proposed NAV page.

7.3 ILS Page

The Instrument Landing System (ILS) page gives information for an instrument landing. The GLIDESLOPE and CDI are used to derive the airplanes position relative to the landing system beams being projected from the threshold of the runway. Fig. 39 shows the proposed ILS page [2]. The tunnel shows the "walls" to stay within so that a landing can be made. As the plane nears the runway, the runway grows and the tunnel shortens until the runway is in sight for the pilot. A trend indicator is proposed in the box below the heading. The differentiated CDI and GLIDESLOPE signals would be used to point the arrow in the correct direction.

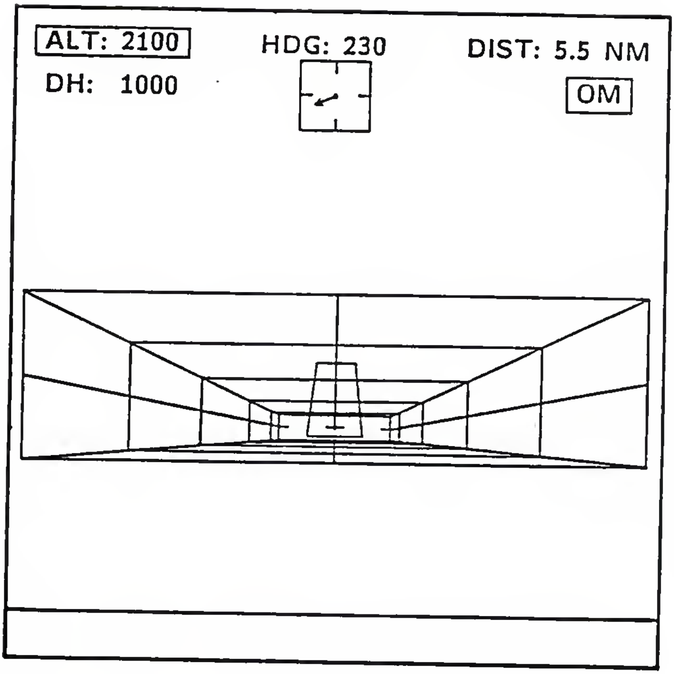

Figure 39. Proposed ILS page.

VIII. CONCLUSIONS

In this thesis, a number of algorithms have been presented that establish communications between an interface and a host computer of an EHSI development system. The algorithms have been implemented on a Zenith 158 personal computer, and the communication routines were interfaced with C. As a result, a system has been designed to operate an EHSI development system. Display update functions have been written by another team member, and the communication routines are used extensively within the functions. The routines seem to be "bug free" and hard to "lock up".

A main program for the system was presented. This program coordinates interrupt vector servicing with the communication routines.

Key routines were presented that service key presses in a calculator-style environment. Throughout these routines, the procedure for using the interface command routines is found.

Several flight simulator signals have been filtered, and useful trend information was obtained from the CDI and GLIDESLOPE signals. The program Flt_ehsi.c was designed so that future project members can sample and filter flight signals.

The research covered in this thesis has contributed to an EHSI Development System which is presently functional. The system display pages are close variations of the ones proposed, and the 2 Hz. update frequency enables real-time viewing of the display pages.

Recommendations for Future Work

There is still much to be done before a prototype can be built and tested. A few suggestions are listed below, and more can be found in [2] and [3].

After a sufficient amount of development and testing of the current display pages, work should be accomplished to determine how the current host computer's work will be accomplished in the cockpit of an airplane. This will most likely entail the design of a dedicated processor board. A processor will be required that enables more than a 2 Hz. screen update rate. The board should be compatible with the DACI, and provisions should be made to make it easy to modify the memory on the board. This would be handy during flight testing. A non-Intel processor will probably be used, and it is hoped that the detailed algorithms presented in this thesis will aid in the writing of code for a different processor.

A very worthwhile project would be a display driver interface that operated on short codes from the host, and translated them to the HP-1345A. This would take many time-

consuming tasks away from the host so that it could increase the update rate, or generate more detailed pages.

The ATC-610 flight simulator has a wealth of information readily avaiable in the form of navigational aids. A data base could be designed that stored information on NDB's, VOR's, and obstacles that are normally presented on flight maps. The main program should then be able to access information on objects that are within a certain radius of the airplane. Using the data base information, the page update functions could then show detailed information of the NDB's, VOR's, and obstacles such as radio towers in the area.

There are many more tasks left to be accomplished before the EHSI is installed in an airplane. For the time being, it is exciting to see one operating from a simulator. With a new set of team members and fresh ideas, it is hoped that the EHSI system can continue to progress towards the cockpit.

REFERENCES

[1] Dyer, S.A., "A Proposed Electronic Horizontal Situation Indicator for use in General-Aviation Aircraft," Proceedings of 1982 Position, Location and Navigation Symposium, pp. 198-205.

[2] Lagerberg, J.D., An Electronic Horizontal Situation Indicator and Development System, M.S. Thesis, Kansas State University, 1987.

[3] Robertson, C., Graphical-page Development of an Electronic Horizontal Situation Indicator, M.S. Thesis, Kansas State University, 1987.

[4] Duncan, R., Advanced MS-DOS, Microsoft Press, 1986.

[5] Hamming, R.W., Digital Filters, Prentice-Hall, Inc., 1977.

[6] Microsoft C Version 4.0 User's Manual, Microsoft Press, 1986.

APPENDIX A

SOURCE CODE

```
PAGE 55,132        ;listing page size is 55 lines by 132 col.
NAME COM_EHSI      ;set name of module
TITLE Communication routines for EHSI development system
;**************************************************************
;*
;* SOURCE FILE:   COM_EHSI.ASM
;*
;* FUNCTION:      Header containing declarations for
;*                communication routines for the EHSI
;*                development system.
;*
;* AUTHOR:        Dave Gruenbacher
;*
;* DATE CREATED:  01Aug86   Version 1.0
;*
;* REVISIONS:     01Dec87
;*                Interfaced with C.
;*                Dave Gruenbacher
;*
;*                01Apr87
;*                Made segment names completely compatible.
;*                Implemented true return of error codes.
;*                Dave Gruenbacher
;*
;**************************************************************

;Segments of the assembly language routines are given the
;same segment names, align types, and combine class as the
;C programs that will be linked with them. This will force
;the assembly language code and C code to be combined so
;that variables can be shared.

_TEXT SEGMENT BYTE PUBLIC 'CODE'
_TEXT ENDS

;declare the C program SCREEN array and data_pkg structure
;as external. The variables are used to get the base
;addresses of the array and structure.
_DATA SEGMENT WORD PUBLIC 'DATA'
EXTRN _SCREEN:word              ;first elelment of SCREEN.
EXTRN _data_pkg:byte            ;first element of data_pkg.
_DATA ENDS

CONST SEGMENT WORD PUBLIC 'CONST'
CONST ENDS

_BSS  SEGMENT WORD PUBLIC 'BSS'
_BSS  ENDS
```

```
NULL  SEGMENT PARA PUBLIC 'BEGDATA'
NULL  ENDS

STACK SEGMENT PARA STACK 'STACK'
STACK ENDS

;All segments except _TEXT are grouped together in a
;small-model C program. _TEXT has its own segment.

DGROUP GROUP CONST, _BSS, _DATA, NULL, STACK

;The default value in the cs register is _TEXT, and the
;default for ds is DGROUP, which as mentioned above is
;the same segment value for several different segments.

ASSUME CS: _TEXT, DS: DGROUP, SS: DGROUP, ES: DGROUP


;Definitions of port addresses, output states, error codes,
;pulse lengths, and time-out delays.

;port addresses.
DATA_PORT_ADDR      EQU    378H
ACK_PORT_ADDR       EQU    379H
STROBE_PORT_ADDR    EQU    37AH
ENABLE_PORT_ADDR    EQU    37AH

;bytes to output to control state of data bus.
ENABLE_READ_DATA    EQU    0FEH
ENABLE_WRITE_DATA   EQU    0FCH
DISABLE_DATA_PORT   EQU    0F4H

;command interrupt vectors.
CMD1_INT_VECTOR   EQU    001H
CMD2_INT_VECTOR   EQU    002H
CMD3_INT_VECTOR   EQU    003H
CMD4_INT_VECTOR   EQU    004H

;error definitions.
NUM_FAULT_ERR       EQU    0D0H

NO_ERROR            EQU    000H

INT_LOW_ACK_ERR     EQU    0E0H
INT_HIGH_ACK_ERR    EQU    0E1H
INT_BUS_CONFLICT    EQU    0E2H

ACK_LOW_ERR         EQU    0F0H
ACK_HIGH_ERR        EQU    0F1H
```

```
;number of entries in data package.
DPKG_ENTRIES        EQU   018H

;end of SCREEN array signal.
END_SCREEN          EQU   0FFH

;pulse widths and time-out values.
INSHAKE_WAIT_LOW    EQU   050H
INSHAKE_WAIT_HIGH   EQU   007H
INSHAKE_LINE_HIGH   EQU   010H
OUTSHAKE_DELAY      EQU   00FH
OUTSHAKE_HOLD_LOW   EQU   00BH
INT_WAIT_LOW        EQU   020H
INT_WAIT_HIGH       EQU   007H

_TEXT     SEGMENT
```

```
SUBTTL GET_DATA_PACKAGE.ASM
PAGE+
;*************************************************************
;*
;* SOURCE_FILE:  COM_EHSI.ASM
;*
;* FUNCTION:     GET_DATA_PACKAGE
;*
;* DESCRIPTION:  This procedure is used to receive the
;*               current data package from the DACI.
;*
;* EXTERNAL
;* VARIABLES:
;*
;*     data_pkg  (structure)
;*               C program structure where the contents of
;*               the received data structure are to be
;*               placed.
;*
;* RETURN
;* REGISTER:
;*
;*     ax        (integer)
;*               error status according to the following:
;*                 NO_ERROR:         Normal completion.
;*                 INT_LOW_ACK_ERR:  DACI did not
;*                                   acknowledge interrupt.
;*                 INT_HIGH_ACK_ERR: DACI did not bring
;*                                   ack line back high
;*                                   after interrupt.
;*                 ACK_LOW_ERR:      Acknowledge line did
;*                                   not go low.
;*                 ACK_HIGH_ERR:     Acknowledge line did
;*                                   not go back high after
;*                                   going low.
;*                 other:            Other error code.
;*
;*
;* REGISTERS
;* CHANGED:      ax,cx,dx
;*
;* FUNCTIONS
;* CALLED:       output_int_byte_proc
;*               inshake_proc
;*               outshake_proc
;*               input_byte_proc
;*               output_byte_proc
;*
;* AUTHOR:       Dave Gruenbacher
;*
```

```
;* DATE CREATED: 01Aug86   Version 1.0
;*
;* REVISIONS:
;*
;*               21Feb87   Version 2.0
;*               overhauled error return and type structure.
;*               Dave Gruenbacher
;*
;*************************************************************

PUBLIC _GET_DATA_PACKAGE
_GET_DATA_PACKAGE proc near

enter_get_data_pkg:
      push bp
      mov  bp,sp
      sub  sp,8
      push di
      push si                       ;save C program registers.

try_cmd_1:
      mov  di,offset ds:_data_pkg  ;get start addr of
                                    ;data package structure.
      mov  al,CMD1_INT_VECTOR       ;get_data_pkg is cmd #1.
      call output_int_byte_proc     ;interrupt the DACI.
      test ax,NO_ERROR              ;check for error.
      jnz  dpkg_error               ;jump if error occurred.

      call inshake_proc             ;wait for byte-ready sig.
      test ax,NO_ERROR              ;check for error.
      jnz  dpkg_error               ;jump if error occurred.

      call input_byte_proc          ;get number of data
                                    ;package entries.
      mov  ah,00H
      mov  cx,ax                    ;num. of entries is in cx.
      cmp  cx,DPKG_ENTRIES          ;cx should be 24D.
      jnz  bad_num_received         ;jump if 24 not received.

read_loop:
      push cx                       ;save cx for later.
      call inshake_proc             ;wait for ack.
      cmp  ax,NO_ERROR              ;check for error.
      jnz  dpkg_error               ;jump if error occurred.
      call input_byte_proc          ;read the data port.
      mov  byte ptr ds:[di],al      ;insert item in data pkg.
      inc  di                       ;point to next data item.
      pop  cx                       ;get counter.
      dec  cx                       ;decrement counter.
      jz   no_dpkg_error            ;jump if done.
```

```
        jmp   read_loop                    ;if not done,
                                           ;get next byte.

bad_num_received:
        sti                                ;turn interrupts back on.
        mov   ax,NUM_FAULT_ERR             ;return error code.
        jmp   exit_get_data_pkg            ;go to exit sequence.

dpkg_error:                                ;return ack error code.
        sti                                ;turn interrupts back on.
        jmp   exit_get_data_pkg            ;go to exit sequence.

no_dpkg_error:
        mov   ax,NO_ERROR                  ;return no error code.
        jmp   exit_get_data_pkg            ;go to exit sequence.

exit_get_data_pkg:
        pop   si                           ;restore C program
        pop   di                           ;registers.
        mov   sp,bp
        pop   bp

        sti                                ;turn interrupts back on.
        ret                                ;return to calling
                                           ;C program.
_GET_DATA_PACKAGE endp

;****end of GET_DATA_PACKAGE********************************
```

```
SUBTTL SEND_SCREEN.ASM
PAGE+
;*****************************************************************
;*
;* SOURCE_FILE:  COM_EHSI.ASM
;*
;* FUNCTION:     SEND_SCREEN
;*
;* DESCRIPTION:  This procedure is used to send the contents
;*               of the SCREEN array to the HP-1345A through
;*               the DACI via command #2.
;*
;*
;* EXTERNAL
;* VARIABLES:
;*
;*     SCREEN    (word)
;*               C program array that holds the words
;*               to be sent.  The end of the array is
;*               detected when the MSB of a word is FFH.
;*
;* RETURN
;* REGISTER:
;*
;*     ax        (integer)
;*               error status according to the following:
;*                 NO_ERROR:          Normal completion.
;*                 INT_LOW_ACK_ERR:   DACI did not
;*                                    acknowledge interrupt.
;*                 INT_HIGH_ACK_ERR:  DACI did not bring
;*                                    ack line back high
;*                                    after interrupt.
;*                 ACK_LOW_ERR:       Acknowledge line did
;*                                    not go low.
;*                 ACK_HIGH_ERR:      Acknowledge line did
;*                                    not go back high after
;*                                    going low.
;*                 other:             Other error code.
;*
;*
;*
;* REGISTERS
;* CHANGED:      ax,cx,dx
;*
;* FUNCTIONS
;* CALLED:       output_int_byte_proc
;*               inshake_proc
;*               outshake_proc
;*               input_byte_proc
;*               output_byte_proc
```

```
;*
;* AUTHOR:        Dave Gruenbacher
;*
;* DATE CREATED: 01Aug86   Version 1.0
;*
;* REVISIONS:
;*
;*               21Feb87   Version 2.0
;*               overhauled error return and
;*               error type structure
;*               Dave Gruenbacher
;*
;*               11Apr87
;*               changed name from UPDATE_SCREEN
;*               to SEND_SCREEN.
;*               Dave Gruenbacher
;*
;************************************************************

PUBLIC _SEND_SCREEN
_SEND_SCREEN PROC NEAR

enter_send_screen:
      push bp                        ;save C program
      mov  bp,sp                     ;registers.
      sub  sp,8
      push di
      push si

      mov  al,CMD2_INT_VECTOR        ;get interrupt vector #2.
      call output_int_byte_proc      ;interrupt the DACI.
      test ax,NO_ERROR               ;error code returned in ax.
      jnz  send_error                ;jump if error occurred.

      mov  di,offset ds:_SCREEN      ;set pointer to array start

send_screen:
      mov  ax,word ptr ds:[di]       ;get SCREEN word.
      push ax                        ;save for sake of LSB.
      xchg al,ah                     ;MSB -> al, LSB -> ah.
      call output_byte_proc          ;send MSB to DACI.
      cmp  ax,NO_ERROR               ;check for error.
      jnz  send_error                ;jump if error occurred
      pop  ax                        ;get original word.
      xchg al,ah                     ;MSB -> al, LSB -> ah.
      cmp  al,END_SCREEN             ;is MSB = FFH?
      jz   no_scr_error              ;jump if finished.
      xchg al,ah                     ;MSB -> ah, LSB -> al.
      call output_byte_proc          ;send LSB to DACI.
      cmp  ax,NO_ERROR               ;check for error.
```

```
        jnz   send_error                    ;jump if error occurred.
        inc   di                            ;increment pointer twice to
        inc   di                            ;point to next SCREEN word.
        jmp   send_screen                   ;jump to send another word.

send_error:                                 ;error code is in ax.
        sti                                 ;turn interrupts back on.
        jmp   exit_send_screen              ;go to exit sequence.

no_scr_error:
        mov   ax,NO_ERROR                   ;return no error occurred.
        jmp   exit_send_screen              ;go to exit sequence.

exit_send_screen:
        pop   si                            ;restore C program
        pop   di                            ;registers.
        mov   sp,bp
        pop   bp

        sti                                 ;turn interrupts back on.
        ret                                 ;return to calling routine.
_SEND_SCREEN endp

;****end of SEND_SCREEN************************************
```

```
SUBTTL RETRIEVE_SCREEN.ASM
PAGE+
;****************************************************************
;*
;* SOURCE_FILE:  COM_EHSI.ASM
;*
;* FUNCTION:     RETRIEVE_SCREEN
;*
;* DESCRIPTION:  This procedure is used to retrieve contents
;*               of a specified block of HP-1345A memory.
;*               The start of the block is taken from
;*               SCREEN[0], and the end is taken from
;*               SCREEN[1]. The retrieved memory is put in
;*               the SCREEN[] array.
;*
;* EXTERNAL
;* VARIABLES:
;*
;*     SCREEN    (word)
;*               On entry, the first and second elements
;*               contain the start and end addresses of the
;*               HP-1345A memory to retrieve, respectively.
;*               On exit, the array holds the memory
;*               recieved through the DACI.
;*
;*
;* RETURN
;* REGISTER:
;*
;*     ax        (integer)
;*               error status according to the following:
;*                 NO_ERROR:          Normal completion.
;*                 INT_LOW_ACK_ERR:   DACI did not
;*                                    acknowledge interrupt.
;*                 INT_HIGH_ACK_ERR:  DACI did not bring
;*                                    ack line back high
;*                                    after interrupt.
;*                 ACK_LOW_ERR:       Acknowledge line did
;*                                    not go low.
;*                 ACK_HIGH_ERR:      Acknowledge line did
;*                                    not go back high after
;*                                    going low.
;*                 other:             Other error code.
;*
;*
;*
;* REGISTERS
;* CHANGED:      ax,bx,cx,dx
;*
;* FUNCTIONS
```

```
;* CALLED:         output_int_byte_proc
;*                inshake_proc
;*                outshake_proc
;*                input_byte_proc
;*                output_byte_proc
;*
;* AUTHOR:        Dave Gruenbacher
;*
;* DATE CREATED: 01Nov86   Version 1.0
;*
;* REVISIONS:
;*
;*                21Feb87   Version 2.0
;*                overhauled error return and
;*                error type structure
;*
;***********************************************************

PUBLIC _RETRIEVE_SCREEN
_RETRIEVE_SCREEN PROC NEAR

        push bp                         ;save C program registers.
        mov  bp,sp
        sub  sp,8
        push di
        push si

        mov  al,CMD3_INT_VECTOR         ;get interrupt vector #3.
        call output_int_byte_proc       ;interrupt the DACI.
        test ax,NO_ERROR                ;ax=0 means no error.
        jnz  send_error                 ;jump if error occurred.

        mov  di,offset ds:_SCREEN       ;point to start of SCREEN.

        mov  ax,word ptr ds:[di]        ;get start address.
        push ax                         ;save start address.
        push ax                         ;save LSB for later.
        xchg al,ah                      ;MSB -> al, LSB -> ah.
        call output_byte_proc           ;send start address MSB.
        test ax,NO_ERROR                ;ax=0 means no error.
        jnz  rtr_error                  ;jump if error occurred.
        pop  ax                         ;get LSB of start address.
        call output_byte_proc           ;send start address LSB.
        test ax,NO_ERROR                ;ax=0 means no error.
        jnz  rtr_error                  ;jump if error occurred.

        inc  di                         ;point to the end address
        inc  di                         ;of the transfer.

        mov  ax,word ptr ds:[di]        ;get end address.
```

```
        push ax                         ;save end address for sub.
        push ax                         ;save LSB for later.
        xchg al,ah                      ;MSB -> al, LSB -> ah.
        call output_byte_proc           ;send end address MSB.
        test ax,NO_ERROR                ;ax=0 means no error.
        jnz  rtr_error                  ;jump if error occurred.
        pop  ax                         ;get LSB of end address.
        call output_byte_proc           ;send end address LSB.
        test ax,NO_ERROR                ;ax=0 means no error.
        jnz  rtr_error                  ;jump if error occurred.

        pop  bx                         ;put end address in bx.
        pop  cx                         ;put start address in cx.
        sub  bx,cx                      ;bx = end - start address

        mov  di,offset ds:_SCREEN       ;point to start of SCREEN.

rtr_screen:
        push bx                         ;save counter.
        call inshake_proc               ;wait for byte ready sig.
        test ax,NO_ERROR                ;check for error.
        jnz  rtr_error                  ;jump if error occurred.
        call input_byte_proc            ;get the MSB of scrn word.
        mov  byte ptr ds:[di+1],al      ;put MSB in screen array.
        inc  di                         ;point to next scrn byte.

        call inshake_proc               ;wait for byte ready sig.
        test ax,NO_ERROR                ;check for error.
        jnz  rtr_error                  ;jump if error occurred.
        call input_byte_proc            ;get the LSB of scrn word.
        mov  byte ptr ds:[di-1],al      ;put LSB in screen array.

        pop  bx                         ;get counter.
        cmp  bx,0                       ;is bx = 0 ?
        jz   rtr_done                   ;jump if done.

        dec  bx                         ;adjust counter.
        inc  di                         ;point to next screen byte.
        jmp  rtr_screen                 ;jump to get another word.

rtr_error:
        sti                             ;turn interrupts back on.
        jmp  exit_rtr_screen            ;jump to exit sequence.

rtr_done:
        mov  ax,NO_ERROR                ;return no error code.
        jmp  exit_rtr_screen            ;jump to exit sequence.

exit_rtr_screen:
        pop  si                         ;restore C program
```

```
      pop   di                          ;registers.
      mov   sp,bp
      pop   bp

      sti                               ;turn interrupts back on.
      ret                               ;return to calling routine.
_RETRIEVE_SCREEN endp

;****end of RETRIEVE_SCREEN*********************************
```

```
SUBTTL TOGGLE_ALARM_SWITCH.ASM
PAGE+
;*****************************************************************
;*
;* SOURCE_FILE:  COM_EHSI.ASM
;*
;* FUNCTION:     TOGGLE_ALARM_SWITCH
;*
;* DESCRIPTION:  This procedure is used to toggle the state
;*               of the alarm on the interface. If an error
;*               occurs, an error code is returned.
;*
;* EXTERNAL
;* VARIABLES:    None.
;*
;* RETURN
;* REGISTER:
;*
;*      ax       (integer)
;*               error status according to the following:
;*                 NO_ERROR:         Normal completion.
;*                 INT_LOW_ACK_ERR:  DACI did not
;*                                   acknowledge interrupt.
;*                 INT_HIGH_ACK_ERR: DACI did not bring
;*                                   ack line back high
;*                                   after interrupt.
;*
;*
;* REGISTERS
;* CHANGED:      ax,cx,dx
;*
;* FUNCTIONS
;* CALLED:       output_int_byte_proc
;*
;* AUTHOR:       Dave Gruenbacher
;*
;* DATE CREATED: 01Aug86   Version 1.0
;*
;* REVISIONS:
;*
;*               21Feb87   Version 2.0
;*               overhauled error return and type structure.
;*               Dave Gruenbacher
;*
;*               11Apr87
;*               changed error return.
;*               Dave Gruenbacher
;*****************************************************************

PUBLIC _TOGGLE_ALARM_SWITCH
```

```
_TOGGLE_ALARM_SWITCH PROC NEAR

enter_tog_alarm:
      push bp                      ;save C program registers.
      mov  bp,sp
      sub  sp,8
      push di
      push si

      mov  al,CMD4_INT_VECTOR      ;get interrupt vector #4.
      call output_int_byte_proc    ;interrupt the DACI.
                                   ;correct error code
                                   ;is in ax for return.
exit_tog_alarm:
      pop  si                      ;restore C program
      pop  di                      ;registers.
      mov  sp,bp
      pop  bp

      sti                          ;turn interrupts back on.
      ret                          ;return to calling routine.
_TOGGLE_ALARM_SWITCH ENDP

;****end of TOGGLE_ALARM_SWITCH*****************************
```

```
SUBTTL INSHAKE_PROC.ASM
PAGE+
;************************************************************
;*
;* SOURCE_FILE:  COM_EHSI.ASM
;*
;* FUNCTION:     INSHAKE_PROC
;*
;* DESCRIPTION:  This procedure waits for the DACI to pulse
;*               the Z-158 INSHAKE line from high to low and
;*               then back to high.
;*
;* ARGUMENTS:    None.
;*
;* RETURN:
;*
;*    ax         error status according to the following:
;*                 NO_ERROR:          Normal completion.
;*                 ACK_LOW_ERR:       Acknowledge line did
;*                                    not go low.
;*                 ACK_HIGH_ERR:      Acknowledge line did
;*                                    not go back high after
;*                                    going low.
;*
;* REGISTERS
;* CHANGED:      ax,cx,dx
;*
;* FUNCTIONS
;* CALLED:       None.
;*
;* AUTHOR:       Dave Gruenbacher
;*
;* DATE CREATED: 01Aug86   Version 1.0
;*
;* REVISIONS:
;*
;*               21Feb87   Version 2.0
;*               tightened time delays and inserted error
;*               handling capabilities.
;*
;************************************************************

inshake_proc proc near

      mov  dx,ACK_PORT_ADDR       ;get inshake port address.
      mov  cx,INSHAKE_WAIT_LOW    ;initialize time-out delay

ack_low_loop:
      in   al,dx                  ;read the inshake port.
      dec  cx
```

```
        jz    low_ack_err                  ;jump if time-out occurred.
        and   al,INSHAKE_LINE_HIGH         ;check if inshake is high.
        jnz   ack_low_loop                 ;if still high, try again.

        mov   cx,INSHAKE_WAIT_HIGH         ;initialize time-out delay

ack_high_loop:
        in    al,dx                        ;read the inshake port.
        dec   cx
        jz    high_ack_err                 ;jump if time-out occurred.
        and   al,INSHAKE_LINE_HIGH         ;check if inshake is high.
        jz    ack_high_loop                ;if still low, try again.

        mov   ax,NO_ERROR                  ;return no error.
        jmp   end_inshake                  ;jump to end.

low_ack_err:
        mov   ax,ACK_LOW_ERR               ;inshake did not go low.
        jmp   end_inshake                  ;jump to end.

high_ack_err:
        mov   ax,ACK_HIGH_ERR              ;inshake did not go high.
        jmp   end_inshake                  ;jump to end.

end_inshake:
        ret                                ;return to calling routine.

inshake_proc endp

;****end of inshake_proc************************************
```

```
SUBTTL OUTSHAKE_PROC.ASM
PAGE+
;****************************************************************
;*
;* SOURCE_FILE:  COM_EHSI.ASM
;*
;* FUNCTION:     OUTSHAKE_PROC
;*
;* DESCRIPTION:  This procedure puts a low pulse on the
;*               Z-158 OUTSHAKE line. There is an
;*               adjustable delay before the pulse is
;*               sent, and the length of the pulse is
;*               also adjustable.
;*
;* ARGUMENTS:    None.
;*
;* RETURN:       None.
;*
;* REGISTERS
;* CHANGED:      al,cx,dx
;*
;* FUNCTIONS
;* CALLED:       None.
;*
;* AUTHOR:       Dave Gruenbacher
;*
;* DATE CREATED: 01Aug86   Version 1.0
;*
;* REVISIONS:    None.
;*
;****************************************************************

outshake_proc proc near
      mov   dx,STROBE_PORT_ADDR    ;get outshake port address.
      in    al,dx                  ;get current ouput value.
      or    al,01H                 ;set the outshake line.

      mov   cx,OUTSHAKE_DELAY      ;need to wait before
os_delay:                          ;pulsing the outshake line
      dec   cx                     ;so that the DACI will be
      jnz   os_delay               ;ready.

      out   dx,al                  ;outshake line is now low.
      dec   al                     ;prepare to bring
                                   ;outshake back high.

      mov   cx,OUTSHAKE_HOLD_LOW   ;need to hold the outshake
os_low_delay:                      ;line low long enough for
      dec   cx                     ;the DACI to see it.
      jnz   os_low_delay
```

```
      out  dx,al                      ;outshake line is now high.
      ret                             ;return to callin routine.
outshake_proc endp

;****end of outshake_proc***********************************
```

```
SUBTTL INPUT_BYTE_PROC.ASM
PAGE+
;*************************************************************
;*
;* SOURCE_FILE:  COM_EHSI.ASM
;*
;* FUNCTION:     INPUT_BYTE_PROC
;*
;* DESCRIPTION:  This procedure is used to read a byte from
;*               the data bus.
;*
;* ARGUMENTS:    None.
;*
;* RETURN:
;*
;*    al         byte read through data port.
;*
;* REGISTERS
;* CHANGED:      ax,cx,dx
;*
;* FUNCTIONS
;* CALLED:       outshake_proc
;*
;* AUTHOR:       Dave Gruenbacher
;*
;* DATE CREATED: 01Aug86   Version 1.0
;*
;* REVISIONS:    None.
;*
;*************************************************************

input_byte_proc proc near
      mov  dx,ENABLE_PORT_ADDR
      mov  al,ENABLE_READ_DATA    ;enable data bus
      out  dx,al                  ;for a read.

      mov  dx,DATA_PORT_ADDR      ;read the data bus and put
      in   al,dx                  ;the result into ax.
      mov  ah,OOH                 ;clear ah.
      push ax                     ;save the input byte.

      mov  dx,STROBE_PORT_ADDR
      mov  al,DISABLE_DATA_PORT
      out  dx,al                  ;disable the data bus.

      call outshake_proc          ;tell DACI the byte
                                  ;was received.
      pop  ax                     ;return the input byte
                                  ;in ax.
      ret                         ;return to calling routine.
```

```
input_byte_proc endp

;****end of input_byte_proc*********************************
```

```
SUBTTL OUTPUT_BYTE_PROC.ASM
PAGE+
;***************************************************************
;*
;* SOURCE_FILE:  COM_EHSI.ASM
;*
;* FUNCTION:     OUTPUT_BYTE_PROC
;*
;* DESCRIPTION:  This procedure is used to output a byte to
;*               the data port. An error is returned if an
;*               acknoweledge was not received.
;*
;* ARGUMENTS:
;*
;*     al        contains the byte to send.
;*
;* RETURN:
;*
;*     ax        error status according to the following:
;*                 NO_ERROR:          Normal completion.
;*                 ACK_LOW_ERR:       Acknowledge line did
;*                                    not go low.
;*                 ACK_HIGH_ERR:      Acknowledge line did
;*                                    not go back high after
;*                                    going low.
;*
;* REGISTERS
;* CHANGED:      ax,cx,dx
;*
;* FUNCTIONS
;* CALLED:       outshake_proc
;*               inshake_proc
;*
;* AUTHOR:       Dave Gruenbacher
;*
;* DATE CREATED: 01Aug86   Version 1.0
;*
;* REVISIONS:    11Apr87
;*               changed error return to ax
;*               Dave Grruenbacher
;*
;***************************************************************

output_byte_proc proc near
      mov  ah,al                  ;save output byte.

      mov  dx,ENABLE_PORT_ADDR
      mov  al,ENABLE_WRITE_DATA
      out  dx,al                  ;enable write to data bus.
```

```
        mov   dx,DATA_PORT_ADDR
        mov   al,ah                        ;put byte to output in al.
        out   dx,al                        ;output byte on data bus.

        call outshake_proc                 ;tell DACI a byte is
                                           ;on the data bus.
        call inshake_proc                  ;wait for ack from DACI,
                                           ;error code returned in ax.
        push ax                            ;save error code.

        mov   dx,STROBE_PORT_ADDR
        mov   al,DISABLE_DATA_PORT
        out   dx,al                        ;disable data bus.

        pop   ax                           ;return error code in ax.
        ret                                ;return to calling routine.
output_byte_proc endp

;****end of output_byte_proc*********************************
```

```
SUBTTL OUTPUT_INT_BYTE_PROC.ASM
PAGE+
;*************************************************************
;*
;* SOURCE_FILE:  COM_EHSI.ASM
;*
;* FUNCTION:     OUTPUT_INT_BYTE_PROC
;*
;* DESCRIPTION:  This procedure is used to output an
;*               interrupt vector to the DACI. If an error
;*               occurs, an error code is returned.
;*
;* ARGUMENTS:
;*
;*     al        contains the interrupt vector to send.
;*
;* RETURN:
;*
;*     ax        error status according to the following:
;*                 NO_ERROR:          Normal completion.
;*                 INT_LOW_ACK_ERR:   DACI did not acknowl-
;*                                    edge the interrupt.
;*                 INT_HIGH_ACK_ERR:  DACI did not bring
;*                                    ack line back high
;*                                    after interrupt.
;*
;* REGISTERS
;* CHANGED:      ax,cx,dx
;*
;* FUNCTIONS
;* CALLED:       None.
;*
;* AUTHOR:       Dave Gruenbacher
;*
;* DATE CREATED: 01Aug86   Version 1.0
;*
;* REVISIONS:
;*
;*               21Feb87   Version 2.0
;*               overhauled error return and error
;*               type structure
;*
;*************************************************************

output_int_byte_proc proc near
int_start:
      mov  ah,al                   ;save interrupt vector.

      mov  dx,ENABLE_PORT_ADDR
      mov  al,ENABLE_WRITE_DATA
```

```
        out   dx,al                      ;enable write to data bus.

        mov   dx,DATA_PORT_ADDR
        mov   al,ah                      ;put interrupt command
                                         ;vector in al.
        out   dx,al                      ;output interrupt vector.

        mov   dx,STROBE_PORT_ADDR        ;get addr. of IRQ_OUT port.
        in    al,dx                      ;get current state of port.
        and   al,11111011B               ;clear IRQ_OUT bit on port.
        out   dx,al                      ;IRQ_OUT line is now low.
        or    al,00000100B               ;reset the IRQ_OUT bit.
        out   dx,al                      ;IRQ_OUT line is back high.

        cli                              ;ignore interrupts for now.

        mov   dx,ACK_PORT_ADDR           ;get ready to read inshake.
        mov   cx,INT_WAIT_LOW            ;wait time for inshake low.

low_int_ack_loop:
        in    al,dx                      ;read the inshake port.
        dec   cx
        jz    no_ack_low                 ;jump if time-out occurred.
        and   al,INSHAKE_LINE_HIGH       ;check if inshake is high.
        jnz   low_int_ack_loop           ;jump if inshake not low.

        mov   cx,INT_WAIT_HIGH           ;wait time for inshake
                                         ;to go high.
high_int_ack_loop:
        in    al,dx                      ;read the inshake port.
        dec   cx
        jz    no_ack_high                ;jump if time-out occurred.
        and   al,INSHAKE_LINE_HIGH       ;check if INSHAKE is high.
        jz    high_int_ack_loop          ;if not high, try again.

        mov   dx,STROBE_PORT_ADDR
        mov   al,ENABLE_READ_DATA        ;enable data bus
        out   dx,al                      ;for a read.

        mov   dx,DATA_PORT_ADDR
        mov   al,00H
        out   dx,al                      ;put 00 on the data bus.

        in    al,dx                      ;read the data bus.
        test  al,00H                     ;is DACI trying to write?
        jnz   int_conflict               ;jump if conflict occurred.

int_okay:
        mov   ax,NO_ERROR                ;return no error in ax.
        jmp   end_int_byte               ;jump to end.
```

```
no_ack_low:
      sti                             ;turn interrupts back on.
      mov   ax,INT_LOW_ACK_ERR        ;return error code in ax.
      jmp   end_int_byte              ;jump to end.

no_ack_high:
      sti                             ;turn interrupts back on.
      mov   ax,INT_HIGH_ACK_ERR       ;return error code in ax.
      jmp   end_int_byte              ;jump to end.

int_conflict:
      sti                             ;turn interrupts back on.
      mov   ax,INT_BUS_CONFLICT       ;return error code in ax.
      jmp   end_int_byte              ;jump to end.

end_int_byte:
      push ax                         ;save error code.

      mov   dx,ENABLE_PORT_ADDR
      mov   al,DISABLE_DATA_PORT
      out   dx,al                     ;disable data port

      pop   ax                        ;return error code in ax.
      ret                             ;return to calling routine.
output_int_byte_proc endp

;****end of output_int_byte_proc****************************

_TEXT ENDS                            ;end of COM_EHSI.ASM code.
END                                   ;end of COM_EHSI.ASM

;****end of COM_EHSI.ASM*************************************
```

```
PAGE 55,132          ;listing page size is 55 lines by 132 col.
NAME INT_EHSI        ;set name of module
TITLE Interrupt routines for the EHSI Development System
;*****************************************************************
;*
;* SOURCE FILE:  INT_EHSI.ASM
;*
;* FUNCTION:     Header containing declarations for
;*               interrupt handling routines for the EHSI
;*               development system.
;*
;* AUTHOR:       Dave Gruenbacher
;*
;* DATE CREATED: 28Dec86   Version 1.0
;*
;* REVISIONS:
;*
;*               01Apr87
;*               Made segment names completely compatible.
;*               Dave Gruenbacher
;*
;*****************************************************************

; Segments of the assembly language routines are given the
; same segment names, align types, and combine class as
; the C programs that will be linked with them. This will
; force the assembly language code and C code to be
; combined so that variables can be shared.

_TEXT SEGMENT BYTE PUBLIC 'CODE'
_TEXT ENDS

_DATA SEGMENT WORD PUBLIC 'DATA'
int_depth  dw   0,0              ;interrupt depth address
int_stack  dw   0,0              ;interrupt stack address
int_OF     dw   0,0              ;old int_Of handler address
_DATA ENDS

CONST SEGMENT WORD PUBLIC 'CONST'
CONST ENDS

_BSS  SEGMENT WORD PUBLIC 'BSS'
_BSS  ENDS

NULL  SEGMENT PARA PUBLIC 'BEGDATA'
NULL  ENDS

STACK SEGMENT PARA STACK 'STACK'
STACK ENDS
```

```
;All segments except _TEXT are grouped together in a small-
;model C program. _TEXT has its own segment.

DGROUP GROUP CONST, _BSS, _DATA, NULL, STACK

;The default value in the cs register is _TEXT, and the
;default for ds is DGROUP, which as mentioned above is
;the same segment value for several different segments.

ASSUME CS: _TEXT, DS: DGROUP, SS: DGROUP, ES: DGROUP


;Definitions of port addresses, output states, error codes,
;pulse lengths, and time-out delays.

;port addresses.
DATA_PORT_ADDR     EQU   378H
ACK_PORT_ADDR      EQU   379H
STROBE_PORT_ADDR   EQU   37AH
ENABLE_PORT_ADDR   EQU   37AH

;bytes to output to control state of data bus.
ENABLE_READ_DATA   EQU   0FEH
ENABLE_WRITE_DATA  EQU   0FCH
DISABLE_DATA_PORT  EQU   0F4H

;offset of arguments in small-model C program.
ARGS               EQU   004H

;maximum depth of interrupts to be stacked.
MAX_DEPTH          EQU   020H

;ehsi shutdown interrupt vector.
STOP_VECTOR        EQU   066H

_TEXT   SEGMENT                        ;start of code.
```

```
SUBTTL INITIALIZE.ASM
PAGE+
;************************************************************
;*
;* SOURCE_FILE:  INT_EHSI.ASM
;*
;* FUNCTION:     INITIALIZE(&int_depth,&int_stack)
;*
;* DESCRIPTION:  Performs initialization of the EHSI
;*               developement system. An interrupt handler
;*               called HANDLER is installed, and the
;*               system interrupt environment is altered
;*               to suit the needs of the main program.
;*               This function is the first function
;*               that the main program should invoke.
;*
;* ARGUMENTS:
;*
;*      &int_depth
;*               is the address of a variable used to tell
;*               the main program how deep the interrupts
;*               are stacked.
;*
;*      &int_stack
;*               is the address of the base of the stack
;*               where interrupt vectors are stored while
;*               the main program cannot keep with the
;*               interrupts from the DACI.
;*
;* RETURN:       None.
;*
;* REGISTERS
;* CHANGED:      ax,cx,dx
;*
;* FUNCTIONS
;* CALLED:       None.
;*
;* AUTHOR:       Dave Gruenbacher
;*
;* DATE CREATED: 29Dec86   Version 1.0
;*
;* REVISIONS:    03Mar87   Version 2.0
;*                         Changed arguments to an interrupt
;*                         stacking format.
;*
;************************************************************

PUBLIC _INITIALIZE
_INITIALIZE proc near
```

```
        push bp                       ;save C program registers.
        mov  bp,sp
        push ds
        push di
        push si

;get the argument &int_depth off the stack, and save it at
;int_depth.
        mov  ax,word ptr [bp+ARGS]   ;pop &int_depth.
        mov  ds:int_depth,ax       ;save offset of int_depth.
        mov  ds:int_depth+2,ds     ;save offset of int_depth.

;get the argument &int_stack off the stack, and save
;it at int_stack.
        mov  ax,word ptr [bp+ARGS+2] ;pop &int_stack.
        mov  ds:int_stack,ax       ;save offset of int_stack.
        mov  ds:int_stack+2,ds     ;save offset of int_stack.


; Get the current address of the handler of interrupt OF, so
; that the address can be saved. The address will be
; reinstalled in RESTORE() before the main program
; terminates. DOS function 35H is used to fetch the current
; address. AL is loaded with the interrupt number(OFH),
; and AH is loaded with the function number(35H) before
; INT 21H is invoked. The function returns the
; segment:offset of interrupt handler in ES:BX respectively.
; The addresses than can be stored in a location
; called "int_Of".

        mov  ax,350FH              ;get ready for function call.
        int  21H                   ;get int OFH handler address.
        mov  ds:int_Of,bx          ;store the offset.
        mov  ds:int_Of+2,es        ;store the segment.


; Install the address of HANDLER as the new address of the
; interrupt OFH interrupt handling routine. DOS function 25H
; is used to accomplish this task. DS:DX is loaded with the
; segment and offset of HANDLER, respectively. AH is loaded
; with the function number(25H), and AL must contain the
; interrupt number(OFH) before INT 21H is invoked.

        push cs
        pop  ds                    ;get the segment of HANDLER.
        mov  dx,offset HANDLER     ;get the offset of HANDLER.
        mov  ax,250FH              ;get ready for function call.
        int  21H                   ;install HANDLER as new -
                                   ;int OF service routine.
```

```
; The interrupt mask register needs to be changed during
; operation of the main program. The system clock interrupt
; must be masked for timing purposes, and the parallel port
; interrupt must be unmasked. The mask register is located
; at port 21H.

        in   al,21H                 ;get current mask register.
        and  al,7FH                 ;unmask interrupt 0FH.
        or   al,01H                 ;mask interrupt 08H.
        out  21H,al                 ;install new mask register.

; Before interrupts can be detected on the parallel port,
; (Int 0F), bit 4 of port 037A must be set high. Bits 5,6,
; and 7 are not used and can also be set to high. Bit 0
; is cleared to initialize the outshake line to high,
; and bit 2 is set so that the 68000 is not interrupted
; prematurely.

        mov  dx,037AH               ;get ready to read port 37AH.
        in   al,dx                  ;get current state of port.
        or   al,11110100B           ;set bits 2,4,5,6, and 7.
        and  al,11111110B           ;clear bit 0.
        out  dx,al                  ;send new state to port 37AH.

; Initialization is complete.

        pop  si                     ;restore C program
        pop  di                     ;registers.
        pop  ds
        pop  bp

        ret                         ;return to calling
                                    ;C routine.
_INITIALIZE endp

;****end of INITIALIZE.ASM***********************************
```

```
SUBTTL HANDLER.ASM
PAGE+
;***************************************************************
;*
;* SOURCE_FILE:   INT_EHSI.ASM
;*
;* FUNCTION:      HANDLER()
;*
;* DESCRIPTION:   The interrupt service routine used to
;*                receive interrupt vectors from the DACI.
;*
;* ARGUMENTS:     None.
;*
;* RETURN:        adjusts the interrupt stack with received
;*                interrupt vectors.
;*
;* REGISTERS
;* CHANGED:       ax,cx,dx
;*
;* FUNCTIONS
;* CALLED:        None.
;*
;* AUTHOR:        Dave Gruenbacher
;*
;* DATE CREATED: 29Dec86   Version 1.0
;*
;* REVISIONS:     03Mar87   Version 2.0
;*                          Changed arguments to an interrupt
;*                          stacking format.
;*
;***************************************************************

HANDLER PROC FAR

      cli                           ;turn interrupts off.

      push ax                       ;save registers used
      push bx                       ;in this routine.
      push cx
      push dx
      push ds
      push si
      push di

      mov  dx,ENABLE_PORT_ADDR
      mov  al,ENABLE_READ_DATA
      out  dx,al                    ;enable data bus for read.

      mov  dx,DATA_PORT_ADDR
      in   al,dx                    ;get the interrupt vector.
```

```
        mov   ah,al                    ;save interrupt vector.

        mov   dx,ENABLE_PORT_ADDR
        mov   al,0F5H                  ;disable port and put
        out   dx,al                    ;OUTSHAKE line low.

        mov   bx,ds:int_depth
        mov   ds,ds:int_depth+2
        mov   cx,word ptr ds:[bx] ;get current depth of the
                                       ;interrupt stack.

        cmp   cx,MAX_DEPTH             ;if the stack is full,
        jge   stack_full               ;jump to outshake delay.

        inc   cx                       ;increment int_depth and
        mov   word ptr ds:[bx],cx ;replace in main C program.
        dec   cx                       ;new vector will be placed at
                                       ;2*(int_depth-1) from base of
                                       ;int_stack.
        mov   al,ah
        mov   ah,0                     ;interrupt vector is in ax.

        cmp   ax,STOP_VECTOR           ;receive the STOP_VECTOR?
        jnz   install                  ;if no, install vector.
        mov   cx,0                     ;if yes, install vector at
                                       ;base of interrupt stack.
        mov   bx,ds:int_depth
        mov   ds,ds:int_depth+2
        mov   word ptr ds:[bx],01 ;set current depth of the
                                       ;interrupt vector stack=1.

install:
        mov   si,cx
        add   si,cx                    ;si = 2*(int_depth-1)

        mov   bx,ds:int_stack          ;get offset of int_stack.
        mov   ds,ds:int_stack+2        ;get segment of int_stack.

; store interrupt vector in interrupt vector stack.
        mov   word ptr ds:[bx+si],ax

; clear the wird abov so C program stops correctly.
        mov   word ptr ds:[bx+si+2],0

        mov   cx,0004H                 ;delay after installation.
        jmp   delay                    ;jump to delay.

stack_full:
        mov   cx,000FH
```

```
delay:
        dec   cx                      ;OUTSHAKE low delay.
        jnz   delay                   ;

        in    al,dx                   ;get state of outshake port.
        dec   al                      ;set outshake line bit.
        out   dx,al                   ;outshake now high.

        mov   al,20H                  ;send EOI signal to
        out   20H,al                  ;8259A PIC.

        pop   di                      ;restore registers saved
        pop   si                      ;on entry.
        pop   ds
        pop   dx
        pop   cx
        pop   bx
        pop   ax

        sti                           ;turn interrupts back on.

        iret                          ;return from interrupt.

HANDLER endp

;****end of HANDLER****************************************
```

```
SUBTTL RESTORE.ASM
PAGE+
;************************************************************
;*
;* SOURCE_FILE:  INT_EHSI.ASM
;*
;* FUNCTION:     RESTORE()
;*
;* DESCRIPTION:  Restores the system environment to its
;*               original state. The interrupt mask register
;*               and the original interrupt 0FH service
;*               routine are restored. This is the last
;*               function that the main program should
;*               call before exiting.
;*
;* ARGUMENTS:    None.
;*
;* RETURN:       None.
;*
;* REGISTERS
;* CHANGED:      ax,cx,dx
;*
;* FUNCTIONS
;* CALLED:       None.
;*
;* AUTHOR:       Dave Gruenbacher
;*
;* DATE CREATED: 29Dec86   Version 1.0
;*
;* REVISIONS:    None.
;*
;************************************************************

PUBLIC _RESTORE
_RESTORE proc near

      push bp                  ;save C program registers.
      mov  bp,sp
      push ds
      push di
      push si

      mov  dx,ds:int_0f        ;get old int 0F offset.
      mov  ds,ds:int_0f+2      ;get old int 0F segment.
      mov  ax,250FH            ;DOS function call prep.
      int  21H                 ;restore old int 0F address.

      in   al,21H              ;get interrupt mask.
      or   al,80H              ;disable parallel port int.
      and  al,0FEH             ;enable system clock int.
```

```
      out  21H,al                 ;install new interrupt mask.

      pop  si                     ;restore C program
      pop  di                     ;registers.
      pop  ds
      pop  bp

      ret                         ;return to C program.

_RESTORE endp

;****end of RESTORE*****************************************

_TEXT   ends

        end

;****end of INT_EHSI.ASM************************************
```

```
/***********************************************************
 *
 *   SOURCE FILE:       ehsi.c
 *
 *
 *   FUNCTION:          ehsi()
 *
 *
 *   DESCRIPTION:       Controls the actions taken on receipt
 *                      of an interrupt vector from the
 *                      interrupt vector stack.
 *                      Iniatialization and restoration are
 *                      also performed from this routine. The
 *                      data package, SCREEN array interrupt
 *                      vector stack, and calculator stack
 *                      are declared within this routine.
 *
 *
 *   DOCUMENTATION
 *   FILES:             None.
 *
 *
 *   ARGUMENTS:         None.
 *
 *
 *   RETURN:            None.
 *
 *
 *   FUNCTIONS
 *   CALLED:            None.
 *
 *
 *   AUTHOR:            Dave Gruenbacher
 *
 *
 *   DATE CREATED:      19Jan87       Version 1.0
 *
 *
 *   REVISIONS:         None.
 *
 *
***********************************************************/
#define  ehsi_main
#include <stdio.h>
#include "data_str.h"

void main()
{
   static unsigned short int_depth = 0, int_stack[90]={0};
   int    page_number = 1, i, int_number;
```

```
char   key_buffer[20];
double x_buffer, y_buffer;

void   INITIALIZE();
void   RESTORE();
void   display_ils_page();
void   display_data_page();
void   display_nav_page();
void   dat_pg_dynamic();
void   nav_pg_dynamic();
void   ils_pg_dynamic();
void   dat_pg_static();
void   nav_pg_static();
void   ils_pg_static();
void   update_key_buffer(),roll_stack();
void   set_altitude();
void   set_estimated_wind();
void   exit();
void   clear_stack();
void   insert_new_freq();
void   set_timer();
void   reset_alarm();
void   do_math();

CLOCK_PKG clock_pkg;
ALARM_PKG alarm_pkg;

clock_pkg.timer_min = 0;
clock_pkg.timer_sec = 0;
clock_pkg.time_out_min = 0;
clock_pkg.time_out_sec = 0;
clock_pkg.adf_freq = 242.0;
clock_pkg.com1_freq = 119.1;
clock_pkg.com2_freq = 121.9;
clock_pkg.vor1_freq = 112.6;
clock_pkg.vor2_freq = 110.1;
clock_pkg.assigned_altitude = 0;
clock_pkg.mda_dh = 0;
clock_pkg.estimated_wind = 0;
clock_pkg.timer_operation_flag = NULL_TIMER;
clock_pkg.timer_status_flag = TIMER_OFF;
clock_pkg.math_operation_flag = 0;

alarm_pkg.airspeed_alarm_flag = ALARM_OFF;
alarm_pkg.assigned_altitude_alarm_flag = ALARM_OFF;
alarm_pkg.mda_dh_alarm_flag = ALARM_OFF;
alarm_pkg.time_out_alarm_flag = ALARM_OFF;

INITIALIZE(&int_depth,int_stack);
key_buffer[0] = '\0';
```

```
    for (;;)
       {
       if (int_depth != 0)
          {
          printf("%X %X\n",int_depth,int_stack[0]);
          int_number = int_stack[0];
          int_depth -= 1;
          for (i=0;i!=int_depth;i++)
             int_stack[i] = int_stack[i+1];

          switch (int_number)
             {
             case 0x60:
                int_number = 0;

                switch (page_number)
                   {
                   case 1:
                      dat_pg_dynamic(&clock_pkg,&alarm_pkg);
                      break;

                   case 2:
                      nav_pg_dynamic(&clock_pkg,&alarm_pkg);
                      break;

                   case 3:
                      ils_pg_dynamic(&clock_pkg,&alarm_pkg);
                      break;

                   default:
                      break;
                   }
                break;

             case 0x65:
                int_number = 0;
                printf("\n\nSYSTEM SWITCH ON.\n\n");
/*              set up static HP-1345A memory                    */
                dat_pg_static();  /* Install data page           */
                for (i=0;i!=100;i++);
                nav_pg_static();  /* Install nav. page           */
                for (i=0;i!=100;i++);
                ils_pg_static();  /* Install ils page            */
                break;

             case 0x66:
                RESTORE();
                printf("\n\nSYSTEM SWITCH OFF.\n\n");
                exit();
```

```
                break;


            default:
                if ((int_number == 0) || (int_number > 0x23))
                break;

                switch (int_number)
                    {
                    case 0x04:    /* 0 received from keypad */
                    case 0x05:    /* . received from keypad */
                    case 0x0A:    /* 1 received from keypad */
                    case 0x0B:    /* 2 received from keypad */
                    case 0x0C:    /* 3 received from keypad */
                    case 0x10:    /* 4 received from keypad */
                    case 0x11:    /* 5 received from keypad */
                    case 0x12:    /* 6 received from keypad */
                    case 0x16:    /* 7 received from keypad */
                    case 0x17:    /* 8 received from keypad */
                    case 0x18:    /* 9 received from keypad */

/*                      clear the buffer if a math operation */
/*                      was just completed.                  */
                        if (clock_pkg.math_operation_flag == 1)
                            {
                            key_buffer[0] = '\0';
                            clock_pkg.math_operation_flag = 0;
                            }

/*                      put key pressed in key_buffer.       */
                        update_key_buffer(int_number,
                                          key_buffer);
                        int_number = 0;
                        break;

                    case 0x02:    /* ENTER hit on keypad    */

                        roll_stack(key_buffer, &x_buffer,
                                   &y_buffer);
                        int_number = 0;
                        break;

                    case 0x08:    /* CLEAR hit on keypad    */

                        int_number = 0;
                        clear_stack(key_buffer, &x_buffer,
                                    &y_buffer);
                        break;

                    case 0x01:    /* new COM1 freq. entered */
```

```
case 0x07:    /* new COM2 freq. entered */
case 0x0D:    /* new VOR1 freq. entered */
case 0x13:    /* new VOR2 freq. entered */
case 0x19:    /* new ADF  freq. entered */

   insert_new_freq(int_number, key_buffer,
                   &clock_pkg);
   int_number = 0;
   break;

case 0x1B:    /* new mda/dh entered     */
case 0x1C:    /* new asgn. alt. entered */

   set_altitude(int_number, key_buffer,
                &clock_pkg);
   int_number = 0;
   break;

case 0x1D:    /* new est. wind entered  */

   int_number = 0;
   set_estimated_wind(key_buffer,
                      &clock_pkg);
   break;

case 0x22:    /* SET TIMER hit          */

   int_number = 0;
   set_timer(key_buffer,&clock_pkg);
   clock_pkg.timer_operation_flag =
                               SET_TIMER;
   break;

case 0x1F:    /* START TIMER hit        */

   int_number = 0;
   clock_pkg.timer_operation_flag =
                             START_TIMER;
   break;

case 0x23:    /* RESET TIMER hit        */

   int_number = 0;
   clock_pkg.timer_operation_flag =
                             RESET_TIMER;
   break;

case 0x1E:    /* SET/RST ALRM hit       */

   int_number = 0;
```

```
                reset_alarm();
                break;

            case 0x03:    /* div  key hit on keypad */
            case 0x09:    /* mult key hit on keypad */
            case 0x0F:    /* add  key hit on keypad */
            case 0x15:    /* sub  key hit on keypad */

                do_math(int_number, key_buffer,
                        &x_buffer, &y_buffer);
                int_number = 0;
                clock_pkg.math_operation_flag = 1;
                break;

            case 0x1A:    /* DAT PAGE key hit       */

                int_number = 0;
                page_number = 1;
                display_data_page();
                break;

            case 0x14:    /* NAV PAGE key hit       */

                int_number = 0;
                page_number = 2;
                display_nav_page();
                break;

            case 0x0E:    /* ILS PAGE key hit       */

                int_number = 0;
                page_number = 3;
                display_ils_page();
                break;

            default:     /* key not implemented.   */
                int_number = 0;
                break;
            }
          break;
        }
      }
    }
}
```

```
/*************************************************************
 *
 *   SOURCE FILE:     data_str.h
 *
 *
 *   FUNCTION:        None.
 *
 *
 *   DESCRIPTION:     This is a file to be included in any
 *                    function that needs to access a member
 *                    of the data package. All the pieces of
 *                    the data package are stored in the
 *                    structure data_pkg. Individual members
 *                    are accessed by using the following
 *                    name: data_pkg.ALTITUDE.
 *
 *
 *   DOCUMENTATION
 *   FILES:           None.
 *
 *
 *   ARGUMENTS:       None.
 *
 *
 *   RETURN:          None.
 *
 *
 *   FUNCTIONS
 *   CALLED:          None.
 *
 *
 *   AUTHOR:          Dave Gruenbacher
 *
 *
 *   DATE CREATED:    22Jan87       Version 1.0
 *
 *
 *   REVISIONS:
 *
 *                    13Apr87       Version 1.1
 *                    Added conditional statements.
 *                    Dave Gruenabcher
 *
*************************************************************/

typedef struct
   {
   unsigned char  BANK              ;
   unsigned char  PITCH             ;
   unsigned char  VERT_SPEED        ;
```

```
    unsigned char   DELTA_X                ;
    unsigned char   DELTA_Y                ;
    unsigned char   MANIFOLD_PRESSURE      ;
    unsigned char   COURSE_DEVIATION       ;
    unsigned char   GLIDESLOPE             ;
    unsigned char   ALTITUDE               ;
    unsigned char   AIRSPEED               ;
    unsigned char   COMPASS                ;
    unsigned char   ADF                    ;
    unsigned char   DME                    ;
    unsigned char   POWER                  ;
    unsigned char   RPM                    ;
    unsigned char   SPARE                  ;
    unsigned char   BINARY_INPUTS          ;
    unsigned char   LAST_KEY               ;
    unsigned char   MONTH                  ;
    unsigned char   DAY                    ;
    unsigned char   DATE                   ;
    unsigned char   HOURS                  ;
    unsigned char   MINUTES                ;
    unsigned char   SECONDS                ;
    } DATA_PKG;


typedef struct
    {
    int     timer_min                      ;
    int     timer_sec                      ;
    int     time_out_min                   ;
    int     time_out_sec                   ;
    int     math_operation_flag            ;
    int     timer_operation_flag           ;
    int     timer_status_flag              ;
    double  adf_freq                       ;
    double  com1_freq                      ;
    double  com2_freq                      ;
    double  vor1_freq                      ;
    double  vor2_freq                      ;
    int     assigned_altitude              ;
    int     mda_dh                         ;
    int     estimated_wind                 ;
    } CLOCK_PKG;

typedef struct
    {
    int     airspeed_alarm_flag            ;
    int     assigned_altitude_alarm_flag;
    int     mda_dh_alarm_flag              ;
    int     time_out_alarm_flag            ;
    } ALARM_PKG;
```

```
#ifdef ehsi_main
DATA_PKG data_pkg = {0};
unsigned short SCREEN[1000] = {0xFFFF};
#else
extern DATA_PKG data_pkg;
extern unsigned short SCREEN[];
extern int GET_DATA_PACKAGE();
extern int SEND_SCREEN();
extern int RETRIEVE_SCREEN();
extern int TOGGLE_ALARM_SWITCH();
#endif


#define  NULL_TIMER    0
#define  START_TIMER   1
#define  RESET_TIMER   2
#define  SET_TIMER     3
#define  TIMER_OFF     0
#define  TIMER_ON      1

#define  ALARM_OFF     0
#define  ALARM_ON      1
```

```
/**************************************************************
 *
 *  SOURCE FILE:        key_buff.c
 *
 *
 *  FUNCTION:           update_key_buffer(key_number,buffer)
 *
 *
 *  DESCRIPTION:        Adds the number pressed on the keypad
 *                      to the current number being shown on
 *                      the command line of the data page.
 *                      This function is called only when
 *                      a number or the decimal point is
 *                      pressed.
 *
 *
 *  DOCUMENTATION
 *  FILES:              None.
 *
 *
 *  ARGUMENTS:
 *
 *      key_number      (unsigned short)
 *                      key pressed on the command keyboard.
 *
 *      buffer          (char *)
 *                      pointer to string being shown on the
 *                      command line of the screen.
 *
 *
 *  RETURN:             None.
 *
 *
 *  FUNCTIONS
 *  CALLED:             SEND_SCREEN()
 *                      string_gen()
 *                      insert()
 *
 *
 *  AUTHOR:             Dave Gruenbacher
 *
 *
 *  DATE CREATED:       20Feb87      Version 1.0
 *
 *
 *  REVISIONS:          12Apr87      Version 2.0
 *                      Changed UPDATE_SCREEN() to
 *                      SEND_SCREEN().
 *
 *
                        Changed error return
```

```
 *                          from SEND_SCREEN().
 *                          Dave Gruenbacher
 *
*************************************************************/
#include "data_str.h"
#include "datpg_xy.h"
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<math.h>

void update_key_buffer(key_number,buffer)
char *buffer;
unsigned short key_number;
{
   int p,length,error;
   float size = 1.5;
   unsigned short conv[30];

   void string_gen();
   int insert();

/* The switch statement decodes the key number into the    */
/* specific character hit. The character is then added on */
/* to the string being displayed on the command line.      */

   switch (key_number)  {
   case 0x04:
       strcat(buffer,"0");            /* 0 key hit          */
       break;

   case 0x05:
       strcat(buffer,".");            /* dec. point key hit */
       break;

   case 0x0A:
       strcat(buffer,"1");            /* 1 key hit          */
       break;

   case 0x0B:
       strcat(buffer,"2");            /* 2 key hit          */
       break;

   case 0x0C:
       strcat(buffer,"3");            /* 3 key hit          */
       break;

   case 0x10:
       strcat(buffer,"4");            /* 4 key hit          */
       break;
```

```
        case 0x11:
            strcat(buffer,"5");             /* 5 key hit          */
            break;

        case 0x12:
            strcat(buffer,"6");             /* 6 key hit          */
            break;

        case 0x16:
            strcat(buffer,"7");             /* 7 key hit          */
            break;

        case 0x17:
            strcat(buffer,"8");             /* 8 key hit          */
            break;

        case 0x18:
            strcat(buffer,"9");             /* 9 key hit          */
            break;

        default:                            /* bad key hit        */
            break;
        }

        p=0;
        SCREEN[p++] = 0xCE00;       /* Point to command line   */
        SCREEN[p++] = 0x7818;       /* screen memory.          */

/* Generate the screen code to show the revised buffer.   */
        string_gen(buffer,command_X0,command_Y0,
                   size,&length,conv);

/* Insert the new screen code into the SCREEN[] array.    */
        p = insert(p,length,conv);

        SCREEN[p++] = 0x8FFF;       /* Jump to end of scr mem. */
        SCREEN[p++] = 0xFFFF;       /* End of SCREEN[] signal. */

/* Send the screen code in SCREEN[] to DACI via cmd #2.   */
        error = 1;
        while (error != 0)
           error = SEND_SCREEN();

        return;
}
```

```
/****************************************************************
 *
 *   SOURCE FILE:        key_entr.c
 *
 *
 *   FUNCTION:           roll_stack(buffer,x,y)
 *
 *
 *   DESCRIPTION:        Replaces the y element of the
 *                       calculator stack with the x element,
 *                       and places the number entered through
 *                       the keypad in the x element. The
 *                       previous y element is lost.
 *                       The commnad line buffer is cleared.
 *
 *
 *   DOCUMENTATION
 *   FILES:              None.
 *
 *
 *   ARGUMENTS:
 *
 *        buffer         (char *)
 *                       pointer to string being shown on the
 *                       command line of the screen.
 *
 *        x              (double *)
 *                       pointer to bottom element of the
 *                       calculator stack.
 *
 *        y              (double *)
 *                       pointer to next to bottom element of
 *                       the calculator stack.
 *
 *
 *   RETURN:             None.
 *
 *
 *   FUNCTIONS
 *   CALLED:             SEND_SCREEN()
 *
 *
 *   AUTHOR:             Dave Gruenbacher
 *
 *
 *   DATE CREATED:       20Feb87       Version 1.0
 *
 *
 *   REVISIONS:          12Apr87       Version 2.0
 *                       Changed UPDATE_SCREEN()
```

```
 *                      to SEND_SCREEN().
 *
 *                      Changed error return
 *                      from SEND_SCREEN().
 *                      Dave Gruenbacher
 *
*************************************************************/
#include "data_str.h"        /* ehsi include file. */
#include<stdlib.h>
#include<string.h>
#include<stdio.h>
#include<math.h>

void roll_stack(buffer,x,y)
char *buffer;
double *x,*y;
{
   int error;

   *y = *x;                  /* replace y with x.              */
   *x = atof(buffer);        /* copy number in buffer to       */
                             /* x element of the stack.        */

   strcpy(buffer,"\0");      /* clear command line buffer.     */

   SCREEN[0] = 0xCE00;       /* point to command line scrn     */
   SCREEN[1] = 0x7818;       /* memory,and place a jump to     */
   SCREEN[2] = 0x8FFF;       /* end to clear the display.      */
   SCREEN[3] = 0xFFFF;       /* end of SCREEN[] for cmd 2.     */

/* Send the SCREEN[] array to the DACI via command #2.         */
   error = 1;
   while (error != 0)
      error = SEND_SCREEN();    /* clear command line.         */

   return;
}
```

```
/**********************************************************************
 *
 *   SOURCE FILE:        key_cler.c
 *
 *
 *   FUNCTION:           clear_stack(buffer,x,y)
 *
 *
 *   DESCRIPTION:        Clears the calculator stack and clears
 *                       the command line buffer.
 *
 *
 *   DOCUMENTATION
 *   FILES:              None.
 *
 *
 *   ARGUMENTS:
 *
 *       buffer          (char *)
 *                       pointer to string being shown on the
 *                       command line of the screen.
 *
 *       x               (double *)
 *                       pointer to bottom element of the
 *                       calculator stack.
 *
 *       y               (double *)
 *                       pointer to next to bottom element of
 *                       the calculator stack.
 *
 *
 *   RETURN:             None.
 *
 *
 *   FUNCTIONS
 *   CALLED:             SEND_SCREEN()
 *                       string_gen()
 *                       insert()
 *
 *
 *   AUTHOR:             Dave Gruenbacher
 *
 *
 *   DATE CREATED:       20Feb87       Version 1.0
 *
 *
 *   REVISIONS:          12Apr87       Version 2.0
 *                       Changed UPDATE_SCREEN()
 *                       to SEND_SCREEN().
 *
```

```
 *                           Changed error return
 *                           from SEND_SCREEN().
 *                           Dave Gruenbacher
 *
*************************************************************/
#include "data_str.h"        /* ehsi include file. */
#include<stdlib.h>
#include<stdio.h>
#include<string.h>

void clear_stack(buffer,x,y)
char *buffer;
double *x,*y;
{
   int error;

   *x = 0.0;                 /* clear x element of stack.  */
   *y = 0.0;                 /* clear y element of stack.  */

   strcpy(buffer,"\0");      /* clear command line buffer. */

   SCREEN[0] = 0xCE00;       /* point to command line scrn */
   SCREEN[1] = 0x7818;       /* memory,and place a jump to */
   SCREEN[2] = 0x8FFF;       /* end to clear the display.  */
   SCREEN[3] = 0xFFFF;       /* end of SCREEN[] for cmd 2. */

/* Send the SCREEN[] array to the DACI via command #2.    */
   error = 1;
   while (error != 0)
      error = SEND_SCREEN();   /* clear command line.     */

   return;
}
```

```
/****************************************************************
 *
 *   SOURCE FILE:          key_math.c
 *
 *
 *   FUNCTION:             do_math(key_number,buffer,x,y)
 *
 *
 *   DESCRIPTION:          Performs either +, -, x, or / on the
 *                         x and y elements of the calculator
 *                         stack. x is taken from the command
 *                         line buffer, and y is the previous
 *                         value of x. The result is then placed
 *                         in x, and the previous element of x
 *                         is put in y. The previous element of
 *                         y is lost. The result is displayed
 *                         on the command line.
 *
 *
 *   DOCUMENTATION
 *   FILES:                None.
 *
 *
 *   ARGUMENTS:
 *
 *        key_number       (unsigned short)
 *                         key pressed on the command keyboard.
 *
 *        buffer           (char *)
 *                         pointer to string being shown on the
 *                         command line of the screen.
 *
 *        x                (double *)
 *                         pointer to bottom element of the
 *                         calculator stack.
 *
 *        y                (double *)
 *                         pointer to next to bottom element of
 *                         the calculator stack.
 *
 *
 *   RETURN:               None.
 *
 *
 *   FUNCTIONS
 *   CALLED:               SEND_SCREEN()
 *                         string_gen()
 *                         insert()
 *
 *
```

```
 *  AUTHOR:            Dave Gruenbacher
 *
 *
 *  DATE CREATED:      20Feb87      Version 1.0
 *
 *
 *  REVISIONS:         12Apr87      Version 2.0
 *                     Changed UPDATE_SCREEN()
 *                     to SEND_SCREEN().
 *
 *                     Changed error return
 *                     from SEND_SCREEN().
 *                     Dave Gruenbacher
 *
*************************************************************/
#define DIVIDE_BY_ZERO_ERROR  1
#include<stdlib.h>
#include<string.h>
#include<stdio.h>
#include<math.h>
#include"datpg_xy.h"      /* data page coordinate file.   */
#include "data_str.h"     /* ehsi include file.           */

void do_math(key_number,buffer,x,y)
unsigned short key_number;     /* key pressed.            */
char *buffer;                  /* string displayed.       */
double *x,*y;                  /* double's on stack.      */
{
   int flag=0,error;
   double z;
   int p,length;
   float size = 1.5;
   unsigned short conv[30];
   void string_gen();
   int insert();

   *y = *x;                 /* roll stack.                */

   *x = atof(buffer);       /* copy number in             */
                            /* buffer to stack.           */

   switch (key_number)
      {
      case 0x03:            /*  find   z = y / x          */
         if (*x != 0.0)
            z = *y / *x;
         else
            flag = DIVIDE_BY_ZERO_ERROR;
         break;
      case 0x09:
```

```
        z = *x * *y;        /*  find   z = x * y             */
        break;

     case 0x0F:             /*  find   z = x + y             */
        z = *x + *y;
        break;

     case 0x15:             /*  find   z = y - x             */
        z = *y - *x;
        break;
     }

  strcpy(buffer,"\0");      /* clear buffer                  */

  if (flag == 0) {
     *y = *x;               /* If no errors occurred, roll   */
     *x = z;                /* stack and place the result    */
     gcvt(z,20,buffer);     /* inthe command line buffer.    */
     }
  else                      /* If there was a divide error,  */
                            /* place an error message in     */
                            /* the command line buffer.      */
     strcpy(buffer,"ZERO DIVIDE ERROR");

  p=0;
  SCREEN[p++] = 0xCE00;     /* point to command line screen  */
  SCREEN[p++] = 0x7818;     /* memory.                       */

  string_gen(buffer,command_X0,command_Y0,
             size,&length,conv);

  p = insert(p,length,conv);

  SCREEN[p++] = 0x8FFF;     /* jump to end of screen mem.    */
  SCREEN[p++] = 0xFFFF;     /* end of SCREEN[] for cmd #2.   */

/* Send the SCREEN[] array to the DACI via command #2.       */
  error = 1;
  while (error != 0)
     error = SEND_SCREEN();   /* send new command line.      */

  return;
}
```

```
/*****************************************************************
 *
 *   SOURCE FILE:        key_dat.c
 *
 *
 *   FUNCTION:           display_data_page()
 *
 *
 *   DESCRIPTION:        Changes the pointer in vector memory
 *                       at address 000H to point to 001H where
 *                       the data page is stored.
 *
 *
 *   DOCUMENTATION
 *   FILES:              None.
 *
 *
 *   ARGUMENTS:          None.
 *
 *
 *   RETURN:             None.
 *
 *
 *   FUNCTIONS
 *   CALLED:             SEND_SCREEN()
 *
 *
 *   AUTHOR:             Dave Gruenbacher
 *
 *
 *   DATE CREATED:       10Feb87       Version 1.0
 *
 *
 *   REVISIONS:          12Apr87       Version 2.0
 *                       Changed UPDATE_SCREEN()
 *                       to SEND_SCREEN().
 *
 *                       Changed error return
 *                       from SEND_SCREEN().
 *                       Dave Gruenbacher
 *
 *
*****************************************************************/
#include <stdio.h>
#include "data_str.h"

void display_data_page()
{
   int  p = 0,error;
```

```
   SCREEN[p++] = 0xC000;      /* Point to screen memory.   */
   SCREEN[p++] = 0x8001;      /* Jump to static dat page.  */
   SCREEN[p++] = 0xFFFF;      /* End of SCREEN[].          */

   error = 1;
   while (error != 0)
      error = SEND_SCREEN(); /* install new jump.          */

   return;
}
```

```
/*************************************************************
 *
 *   SOURCE FILE:        key_nav.c
 *
 *
 *   FUNCTION:           display_nav_page()
 *
 *
 *   DESCRIPTION:        Changes the pointer in vector memory
 *                       at address 000H to point to 300H where
 *                       the nav page is stored.
 *
 *
 *   DOCUMENTATION
 *   FILES:              None.
 *
 *
 *   ARGUMENTS:          None.
 *
 *
 *   RETURN:             None.
 *
 *
 *   FUNCTIONS
 *   CALLED:             SEND_SCREEN()
 *
 *
 *   AUTHOR:             Dave Gruenbacher
 *
 *
 *   DATE CREATED:       10Feb87      Version 1.0
 *
 *
 *   REVISIONS:          12Apr87      Version 2.0
 *                       Changed UPDATE_SCREEN()
 *                       to SEND_SCREEN().
 *
 *                       Changed error return
 *                       from SEND_SCREEN().
 *                       Dave Gruenbacher
 *
 *
*************************************************************/
#include <stdio.h>
#include "data_str.h"

void display_nav_page()
{
   int  p = 0,error;
```

```
    SCREEN[p++] = 0xC000;       /* Point to screen memory.   */
    SCREEN[p++] = 0x8300;       /* Jump to static nav page.  */
    SCREEN[p++] = 0xFFFF;       /* End of SCREEN[].          */

    error = 1;
    while (error != 0)
       error = SEND_SCREEN(); /* install new jump.         */

    return;
}
```

```
/*****************************************************************
 *
 *  SOURCE FILE:       key_ils.c
 *
 *
 *  FUNCTION:          display_ils_page()
 *
 *
 *  DESCRIPTION:       Changes the pointer in vector memory
 *                     at address 000H to point to 500H where
 *                     the ils page is stored.
 *
 *
 *  DOCUMENTATION
 *  FILES:             None.
 *
 *
 *  ARGUMENTS:         None.
 *
 *
 *  RETURN:            None.
 *
 *
 *  FUNCTIONS
 *  CALLED:            SEND_SCREEN()
 *
 *
 *  AUTHOR:            Dave Gruenbacher
 *
 *
 *  DATE CREATED:      10Feb87      Version 1.0
 *
 *
 *  REVISIONS:         12Apr87      Version 2.0
 *                     Changed UPDATE_SCREEN()
 *                     to SEND_SCREEN().
 *
 *                     Changed error return
 *                     from SEND_SCREEN().
 *                     Dave Gruenbacher
 *
 *
*****************************************************************/
#include <stdio.h>
#include "data_str.h"

void display_ils_page()
{
   int  p = 0,error;
```

```
    SCREEN[p++] = 0xC000;       /* Point to screen memory.   */
    SCREEN[p++] = 0x8500;       /* Jump to static ils page.  */
    SCREEN[p++] = 0xFFFF;       /* End of SCREEN[].          */

    error = 1;
    while (error != 0)
       error = SEND_SCREEN(); /* install new jump.         */

    return;
}
```

```
/*****************************************************************
 *
 *   SOURCE FILE:       key_alrm.c
 *
 *
 *   FUNCTION:          reset_alarm()
 *
 *
 *   DESCRIPTION:       Toggles the system alarm by invoking
 *                      interface command #4.
 *
 *
 *   DOCUMENTATION
 *   FILES:             None.
 *
 *
 *   ARGUMENTS:         None.
 *
 *
 *   RETURN:            None.
 *
 *
 *   FUNCTIONS
 *   CALLED:            TOGGLE_ALARM_SWITCH()
 *
 *
 *   AUTHOR:            Dave Gruenbacher
 *
 *
 *   DATE CREATED:      10Feb87       Version 1.0
 *
 *
 *   REVISIONS:         12Apr87       Version 2.0
 *                      Changed error return from
 *                      TOGGLE_ALARM_SWITCH().
 *                      Dave Gruenbacher
 *
 *
*****************************************************************/
#include "data_str.h"
#include<stdio.h>

void reset_alarm()
{
   int error = 1;

/* Call TOGGLE_ALARM_SWITCH to toggle on/off state of         */
/* the DACI alarm.                                            */
   while (error != 0))
      error = TOGGLE_ALARM_SWITCH();    /* do command 4.     */
```

```
    return;
}
```

```
/*************************************************************
 *
 *   SOURCE FILE:       key_cmd3.c
 *
 *
 *   FUNCTION:          call_cmd3(buffer,x,y)
 *
 *
 *   DESCRIPTION:       Key function that uses command #3 to
 *                      retrieve HP-1345A memory. x contains
 *                      the starting address, and y contains
 *                      the ending address. The received code
 *                      is printed on the screen.
 *
 *
 *   DOCUMENTATION
 *   FILES:             None.
 *
 *
 *   ARGUMENTS:
 *
 *       buffer         (char *)
 *                      pointer to string being shown on the
 *                      command line of the screen.
 *
 *       x              (double *)
 *                      pointer to bottom element of the
 *                      calculator stack.
 *
 *       y              (double *)
 *                      pointer to next to bottom element of
 *                      the calculator stack.
 *
 *
 *   RETURN:            None.
 *
 *
 *   FUNCTIONS
 *   CALLED:            None.
 *
 *
 *   AUTHOR:            Dave Gruenbacher
 *
 *
 *   DATE CREATED:      13Apr87      Version 1.0
 *
 *
 *   REVISIONS:         None.
 *
 *
```

```
*************************************************************/
#include<stdlib.h>
#include<string.h>
#include<stdio.h>
#include "data_str.h"

void call_cmd3(buffer,x,y)
char *buffer;                  /* string displayed          */
double *x,*y;                  /* double's on stack         */
{
   int i,num_words,error;

   *y = *x;                    /* roll stack                */

   *x = atof(buffer);          /* copy number in buffer     */
                               /* to calculator stack.      */
   SCREEN[0] = (int)(*y);      /* x contains starting addr. */
   SCREEN[1] = (int)(*x);      /* y contains starting addr. */

/* check for overflow or underflow.                         */
   num_words = SCREEN[1] - SCREEN[0] +1;
   if ((num_words >= 1000)||(num_words < 1))
      return;

/* check for out of bounds.                                 */
   if ((SCREEN[0] < 0)||(SCREEN[1] > 4095))
      return;

   strcpy(buffer,"\0");        /* clear buffer              */

/* RETRIEVE screen memory via command #3.                   */
   error = 1;
   while (error != 0)
      error = RETRIEVE_SCREEN();

/* print the received screen code.                          */
   for (i=0;i<(num_words-1);i++)
      printf("SCREEN[%d] = %x\n",i,SCREEN[i]);

   return;
}
```

```
/*************************************************************
 *
 *  SOURCE FILE:        flt_ehsi.c
 *
 *
 *  FUNCTION:           flt_ehsi()
 *
 *
 *  DESCRIPTION:        This program allows a user to sample
 *                      a number of signals coming from the
 *                      flight simulator, and to also filter
 *                      the incoming stream. The unfiltered
 *                      and filtered data are written to
 *                      user specified filenames after the
 *                      sample is taken.
 *
 *
 *  DOCUMENTATION
 *  FILES:              None.
 *
 *
 *  ARGUMENTS:          None.
 *
 *
 *
 *  RETURN:             None.
 *
 *
 *  FUNCTIONS
 *  CALLED:             INITIALIZE()
 *                      GET_DATA_PACKAGE()
 *                      RESTORE()
 *                      ehsi_filter()
 *
 *
 *  AUTHOR:             Dave Gruenbacher
 *
 *
 *  DATE CREATED:       02Apr87       Version 1.0
 *
 *
 *  REVISIONS:          None.
 *
 *
 *************************************************************/
#include <stdio.h>

typedef struct
   {
   unsigned char  BANK                  ;
```

```
    unsigned char  PITCH                    ;
    unsigned char  VERT_SPEED               ;
    unsigned char  DELTA_X                  ;
    unsigned char  DELTA_Y                  ;
    unsigned char  MANIFOLD_PRESSURE        ;
    unsigned char  COURSE_DEVIATION         ;
    unsigned char  GLIDESLOPE               ;
    unsigned char  ALTITUDE                 ;
    unsigned char  AIRSPEED                 ;
    unsigned char  COMPASS                  ;
    unsigned char  ADF                      ;
    unsigned char  DME                      ;
    unsigned char  POWER                    ;
    unsigned char  RPM                      ;
    unsigned char  SPARE                    ;
    unsigned char  BINARY_INPUTS            ;
    unsigned char  LAST_KEY                 ;
    unsigned char  MONTH                    ;
    unsigned char  DAY                      ;
    unsigned char  DATE                     ;
    unsigned char  HOURS                    ;
    unsigned char  MINUTES                  ;
    unsigned char  SECONDS                  ;
    } DATA_PKG;

DATA_PKG data_pkg = {0};

extern int GET_DATA_PACKAGE( );
unsigned short SCREEN[1];

float data_array[4096], filt_array[4096];

void main()
{
   static unsigned short int_depth = 0, int_stack[10]={0};
   int    index = 0, int_number, i, error;
   int    num_samples = 0,element_num = -1;
   FILE   *stream;
   char   data_file[15],filt_file[15];
   unsigned char *element;

   void   INITIALIZE();
   void   RESTORE();
   float  ehsi_filter();

/* Prompt the user to pick the data package element he    */
/* wishes to sample. The "while" loop ensures a           */
/* valid choice.                                          */
   while ((element_num < 0)||(element_num > 12))
      {
```

```
        printf("\nEnter the number corresponding to the\n");
        printf("data package element you wish to sample:\n");
        printf("     BANK                    - 0 \n");
        printf("     PITCH                   - 1 \n");
        printf("     VERT_SPEED              - 2 \n");
        printf("     DELTA_X                 - 3 \n");
        printf("     DELTA_Y                 - 4 \n");
        printf("     COURSE_DEVIATION        - 6 \n");
        printf("     GLIDESLOPE              - 7 \n");
        printf("     ALTITUDE                - 8 \n");
        printf("     AIRSPEED                - 9 \n");
        printf("     COMPASS                 - 10\n");
        printf("     ADF                     - 11\n");
        printf("     DME                     - 12\n");
        scanf("%d",&element_num);
        }

     element = &data_pkg.BANK +
                element_num*sizeof(unsigned char);

  /* Prompt user to enter number of samples to be taken.    */
     printf("\nEnter number of samples to be taken:\n");
     printf("(4096 max)\n");
     scanf("%d",&num_samples);

  /* Prompt user to enter name of file to place              */
  /* unfiltered data.                                        */
     printf("\nEnter name of file to place raw data: \n");
     scanf("%s",data_file);

  /* Prompt user to enter name of file to place              */
  /* filtered data.                                          */
     printf("\nEnter name of file to place filtered data:\n");
     scanf("%s",filt_file);

     INITIALIZE(&int_depth,int_stack);

  /* Collect the data until num_samples has been taken.     */
     for (;;)
        {
        if (int_depth != 0)
           {
           int_number = int_stack[0];
           int_depth -= 1;
           for (i=0;i!=int_depth;i++)
              int_stack[i] = int_stack[i+1];

           if (int_number == 0x60)
              {
              error = 1;
```

```
            while (error != 0)
               error = GET_DATA_PACKAGE();

            data_array[index] = (float)(*element) * 4;
            printf("%d,%d\n",index,*element);
            filt_array[index++] = ehsi_filter(*element*4);

            if (index == (num_samples+1))
/*             If done, write the two files to the          */
/*             specified names and exit the routine.        */

               {
/*             Write the raw data to file "data_file".      */
               stream = fopen(data_file,"wb");
               index = fwrite((char *)data_array,
                          sizeof(float), num_samples,stream);
               fclose(stream);

/*             Write the filtered data to "filt_file".      */
               stream = fopen(filt_file,"wb");
               index = fwrite((char *)filt_array,
                          sizeof(float), num_samples,stream);
               fclose(stream);

               RESTORE();
               exit(0);
               }
            }

            if (int_number == 0x66)
               {
               RESTORE();
               fclose(stream);
               exit(0);
               }

         }
      }
}
```

```
/***************************************************************
 *
 *   SOURCE FILE:        ehsifilt.c
 *
 *
 *   FUNCTION:           ehsi_filter()
 *
 *
 *   DESCRIPTION:        This program does the actual filtering
 *                       of the input data stream according to
 *                       the values in flt_ehsi.h.
 *
 *
 *   DOCUMENTATION
 *   FILES:              None.
 *
 *
 *   ARGUMENTS:
 *
 *      element          (int)
 *                       new input value.
 *
 *
 *
 *   RETURN:
 *
 *      result           (float)
 *                       result of filter function.
 *
 *
 *   FUNCTIONS
 *   CALLED:             None.
 *
 *
 *   AUTHOR:             Dave Gruenbacher
 *
 *
 *   DATE CREATED:       08Apr87       Version 1.0
 *
 *
 *   REVISIONS:          None.
 *
 *
***************************************************************/
#include "flt_ehsi.h"

float ehsi_filter(element)
int element;
{
   static float data[FILT_ORDER+1];
```

```
   int i;
   float result;

   for (i=FILT_ORDER;i>0;i--)
      data[i] = data[i-1];

   data[0] = (float)(element);

   result = data[0] * COEFF0 +
            data[1] * COEFF1 +
            data[2] * COEFF2 +
            data[3] * COEFF3 +
            data[4] * COEFF4 +
            data[5] * COEFF5 +
            data[6] * COEFF6;

   return(result);
}
```

```
/*****************************************************************
 *
 *   SOURCE FILE:        flt_ehsi.h
 *
 *
 *   FUNCTION:           include file for fd_filt.c
 *
 *
 *   DESCRIPTION:        This include file allows a user to
 *                       change the coefficients and order
 *                       of filter to use in flt_ehsi.c.
 *                       Fd_filt.c is the file that includes
 *                       this file, since the actual
 *                       filter is located in that file.
 *
 *
 *   DOCUMENTATION
 *   FILES:              None.
 *
 *
 *   ARGUMENTS:          None.
 *
 *
 *   RETURN:             None.
 *
 *
 *   FUNCTIONS
 *   CALLED:             None.
 *
 *
 *   AUTHOR:             Dave Gruenbacher
 *
 *
 *   DATE CREATED:       10Apr87      Version 1.0
 *
 *
 *   REVISIONS:          None.
 *
 *
******************************************************************/
#define FILT_ORDER   6

#define COEFF0  -.107
#define COEFF1  -.0714
#define COEFF2  -.0357
#define COEFF3    0
#define COEFF4   .0357
#define COEFF5   .0714
#define COEFF6   .107
```

```
#*****************************************************************
#*
#*   SOURCE FILE:        flt_ehsi
#*
#*
#*   FUNCTION:           MAKE file.
#*
#*
#*   DESCRIPTION:        MAKE description file
#*                       for FLT_EHSI.EXE.
#*
#*
#*   FILES
#*   NEEDED:             flt_ehsi.c
#*                       flt_ehsi.h
#*                       filtehsi.c
#*                       com_ehsi.asm
#*                       int_ehsi.asm
#*
#*
#*   AUTHOR:             Dave Gruenbacher
#*
#*
#*   DATE CREATED:       10Apr87       Version 1.0
#*
#*
#*   REVISIONS:          None.
#*
#*
#****************************************************************/

flt_ehsi.obj : flt_ehsi.c
      msc flt_ehsi;

ehsifilt.obj : ehsifilt.c flt_ehsi.h
      msc ehsifilt;

com_ehsi.obj : com_ehsi.asm
      masm com_ehsi;

int_ehsi.obj : int_ehsi.asm
      masm int_ehsi;

flt_ehsi.exe : flt_ehsi.obj ehsifilt.obj\
               com_ehsi.obj int_ehsi.obj
      link flt_ehsi ehsifilt com_ehsi int_ehsi;
```

```
**************************************************************
*
*   SOURCE FILE:      CONVERT.FOR
*
*   DESCRIPTION:      Converts Z-158 created data file into
*                     the unformatted VAX format so that
*                     the data file can be read by RALPH2.
*
*   ARGUMENTS:        Z-158 data file.
*
*   RETURN:           Unformatted VAX format data file.
*
*   DATE CREATED:     10Mar87
*
*   AUTHOR:           Dave Gruenbacher
*
*   REVISIONS:        None.
*
**************************************************************

      INTEGER*2     A(5000)
      CHARACTER*15 INFILE,OUTFILE

 10   WRITE (*,*) 'Enter name of Z-158 input data:'
      READ (*,15,ERR=10) INFILE
 20   WRITE(*,*) 'Enter name of unformatted output file:'
      READ (*,15,ERR=20) OUTFILE
 15   FORMAT (A15)

      OPEN (UNIT = 1, FILE=INFILE,  STATUS = 'OLD')
      OPEN (UNIT = 2, FILE=OUTFILE, STATUS = 'NEW',
     +      FORM = 'UNFORMATTED')

      NEXT =1
      DO WHILE(.TRUE.)
          READ(1,99,END=100) LEN,(A(I),I=NEXT,NEXT+LEN/2-1)
          NEXT = NEXT + LEN/2
      END DO

 99   FORMAT(Q,5000A2)

 100  WRITE(2) (A(I+1),A(I), I=1,NEXT-1,2)
      TYPE *, NEXT/2
      END
```

APPENDIX B

PROGRAM MAINTENANCE

Compiling and Linking

To create an executable file to operate the EHSI Development System, issue the following command from within the directory containing all the source files:

MAKE EHSI <RETURN>

This will invoke Microsoft's MAKE utility [6]. The required MAKE file is included starting at page 171. MAKE looks at the target file and determines whether a dependent file has been modified. If one has, then the commands on the lines following are performed until a blank line is encountered. The compile and link are performed in the process of executing MAKE.

```
#/*************************************************************
#*
#*  SOURCE FILE:       ehsi
#*
#*
#*  FUNCTION:          None.
#*
#*
#*  DESCRIPTION:       Make file for the EHSI host program.
#*                     Used when running make utility.
#*
#*
#*
#*  DOCUMENTATION
#*  FILES:             None.
#*
#*
#*  ARGUMENTS:         None.
#*
#*
#*  RETURN:            None.
#*
#*
#*  FUNCTIONS
#*  CALLED:            None.
#*
#*
#*  AUTHOR:            Dave Gruenbacher
#*                     Chuck Robertson
#*
#*
#*  DATE CREATED:      10Apr87      Version 1.0
#*
#*
#*  REVISIONS:         None.
#*
#*
#************************************************************/


insert.obj   : insert.c
       msc insert;
       lib ehsi-+insert,ehsi.crs;

time_gen.obj : time_gen.c
       msc time_gen;
       lib ehsi-+time_gen,ehsi.crs;

line.obj     : line.c
       msc line;
```

```
        lib ehsi-+line,ehsi.crs;

clim_box.obj : clim_box.c
        msc clim_box;
        lib ehsi-+clim_box,ehsi.crs;

clim_hsh.obj : clim_hsh.c
        msc clim_hsh;
        lib ehsi-+clim_hsh,ehsi.crs;

climrate.obj : climrate.c
        msc climrate;
        lib ehsi-+climrate,ehsi.crs;

climfilt.obj : climfilt.c climfilt.h
        msc climfilt;
        lib ehsi-+climfilt,ehsi.crs;

arc_circ.obj : arc_circ.c
        msc arc_circ;
        lib ehsi-+arc_circ,ehsi.crs;

str_gen.obj  : str_gen.c
        msc str_gen;
        lib ehsi-+str_gen,ehsi.crs;

plane.obj    : plane.c
        msc plane;
        lib ehsi-+plane,ehsi.crs;

waypoint.obj : waypoint.c
        msc waypoint;
        lib ehsi-+waypoint,ehsi.crs;

vortac.obj   : vortac.c
        msc vortac;
        lib ehsi-+vortac,ehsi.crs;

box.obj      : box.c
        msc box;
        lib ehsi-+box,ehsi.crs;

compass.obj  : compass.c
        msc compass;
        lib ehsi-+compass,ehsi.crs;

clim_aro.obj : clim_aro.c
        msc clim_aro;
        lib ehsi-+clim_aro,ehsi.crs;
```

```
arrow.obj    : arrow.c
       msc arrow;
       lib ehsi-+arrow,ehsi.crs;

ndb.obj      : ndb.c
       msc ndb;
       lib ehsi-+ndb,ehsi.crs;

heading.obj  : heading.c
       msc heading;
       lib ehsi-+heading,ehsi.crs;

runway.obj   : runway.c
       msc runway;
       lib ehsi-+runway,ehsi.crs;

zero_pad.obj : zero_pad.c
       msc zero_pad;
       lib ehsi-+zero_pad,ehsi.crs;

timer.obj    : timer.c data_str.h
       msc timer;
       lib ehsi-+timer,ehsi.crs;

altitude.obj : altitude.c data_str.h
       msc altitude;
       lib ehsi-+altitude,ehsi.crs;

dme.obj      : dme.c data_str.h
       msc dme;
       lib ehsi-+dme,ehsi.crs;

ndb_angl.obj : ndb_angl.c
       msc ndb_angl;
       lib ehsi-+ndb_angl,ehsi.crs;

airspeed.obj : airspeed.c data_str.h
       msc airspeed;
       lib ehsi-+airspeed,ehsi.crs;

climrate.obj : climrate.c  data_str.h
       msc climrate;
       lib ehsi-+climrate,ehsi.crs;

ils_cmps.obj : ils_cmps.c
       msc ils_cmps;
       lib ehsi-+ils_cmps,ehsi.crs;
```

```
hdg_brg.obj  : hdg_brg.c
       msc hdg_brg;
       lib ehsi-+hdg_brg,ehsi.crs;

key_alrm.obj : key_alrm.c
       msc key_alrm;
       lib key_ehsi-+key_alrm,key_ehsi.crs;

key_dat.obj  : key_dat.c
       msc key_dat;
       lib key_ehsi-+key_dat,key_ehsi.crs;

key_nav.obj  : key_nav.c
       msc key_nav;
       lib key_ehsi-+key_nav,key_ehsi.crs;

key_ils.obj  : key_ils.c
       msc key_ils;
       lib key_ehsi-+key_ils,key_ehsi.crs;

key_entr.obj : key_entr.c
       msc key_entr;
       lib key_ehsi-+key_entr,key_ehsi.crs;

key_cler.obj : key_cler.c
       msc key_cler;
       lib key_ehsi-+key_cler,key_ehsi.crs;

key_freq.obj : key_freq.c data_str.h
       msc key_freq;
       lib key_ehsi-+key_freq,key_ehsi.crs;

key_stmr.obj : key_stmr.c data_str.h
       msc key_stmr;
       lib key_ehsi-+key_stmr,key_ehsi.crs;

key_math.obj : key_math.c
       msc key_math;
       lib key_ehsi-+key_math,key_ehsi.crs;

key_buff.obj : key_buff.c
       msc key_buff;
       lib key_ehsi-+key_buff,key_ehsi.crs;

key_alt.obj  : key_alt.c data_str.h
       msc key_alt;
       lib key_ehsi-+key_alt,key_ehsi.crs;
```

```
key_wind.obj : key_wind.c data_str.h
       msc key_wind;
       lib key_ehsi-+key_wind,key_ehsi.crs;

key_cmd3.obj : key_cmd3.c
       msc key_cmd3;
       lib key_ehsi-+key_cmd3,key_ehsi.crs;

dat_pg_s.obj   : dat_pg_s.c datpg_xy.h
       msc dat_pg_s;

dat_pg_d.obj    : dat_pg_d.c data_str.h datpg_xy.h
       msc dat_pg_d;

nav_pg_s.obj   : nav_pg_s.c navpg_xy.h
       msc nav_pg_s;

nav_pg_d.obj   : nav_pg_d.c data_str.h navpg_xy.h
       msc nav_pg_d;

ils_pg_s.obj   : ils_pg_s.c ilspg_xy.h
       msc ils_pg_s;

ils_pg_d.obj   : ils_pg_d.c data_str.h ilspg_xy.h
       msc ils_pg_d;

int_ehsi.obj : int_ehsi.asm
       masm int_ehsi;

com_ehsi.obj : com_ehsi.asm
       masm com_ehsi;

ehsi.obj     : ehsi.c data_str.h
       msc ehsi;


ehsi.exe : ehsi.obj dat_pg_s.obj nav_pg_s.obj ils_pg_s.obj\
           dat_pg.obj nav_pg_d.obj nav_pg_d.obj\
           com_ehsi.obj int_ehsi.obj ehsi.lib key_ehsi.lib
       link ehsi dat_pg_s nav_pg_s ils_pg_s dat_pg_d
           nav_pg_d ils_pg_d com_ehsi int_ehsi/stack:4000,
           ehsi,ehsi,ehsi.lib key_ehsi.lib;
```

Adjusting Pulse-widths and Time-out delays

Expressions are defined in the header of com_ehsi.asm that control the widths of output pulses and the maximum amount of time to wait for an acknowledge pulse. Each expression is discussed below, and the equations assume that the Z-158 is operating at 8MHz.

INSHAKE_WAIT_LOW controls the maximum amount of time that INSHAKE_PROC will wait for a high-to-low transition on the DACI OUTSHAKE line. The time in microseconds is found from the following equation:

$$\frac{27 + (\text{INSHAKE WAIT LOW} - 1)*34}{8}$$

OUTSHAKE_WAIT_HIGH controls the maximum amount of time that INSHAKE_PROC will wait for the DACI OUTSHAKE line to return back to high after going low. The time in microseconds is found from the following equation:

$$\frac{18 + (\text{INSHAKE WAIT HIGH} - 1)*34}{8}$$

OUTSHAKE_DELAY controls how long OUTSHAKE_PROC waits before actually pulsing the OUTSHAKE line. The timing diagrams in Sec. 2.2 show that the DACI needs time to get ready to watch for a pulse on the Z-158 OUTSHAKE line. The following equation gives the time in microseconds:

$$\frac{43 + (\text{OUTSHAKE DELAY} - 1)*18}{8}$$

OUTSHAKE_HOLD_LOW controls the time that OUTSHAKE_PROC

keeps the Z-158 OUTSHAKE line low. The following equation gives the time in microseconds:

$$\frac{18 + (\text{OUTSHAKE HOLD LOW} - 1)*18}{8}$$

INT_WAIT_LOW controls the length of time that the Z-158 will wait for the DACI to pulse the OUTSHAKE line after being interrupted by OUTPUT_INT_BYTE_PROC. The time in microseconds is given below:

$$\frac{22 + (\text{INT WAIT LOW} - 1)*34}{8}$$

INT_WAIT_HIGH controls the time that OUTPUT_INT_BYTE_PROC waits for the interrupt acknowledge pulse from the DACI to go back high after going low. The time in microseconds is given below:

$$\frac{16 + (\text{INT WAIT HIGH} - 1)*34}{8}$$

APPENDIX C

MODIFICATIONS

Z-158 Parallel Port Modification

The Z-158 parallel port was modified to enable data reads from the data bus. The output latch of port 0378H was always enabled previously, thus only allowing the latch to be read instead of data coming through the external parallel port connection. As can be seen in Fig. 40, the RPA line was inverted and sent to the output latch's enable pin to correct the problem. This forces the output latch to go into the high impedance state while the read is occurring, so that the data on the parallel port is read instead of the latch. The bottom figure is the corrected circuit, and the top diagram shows the original circuit. The modification was accomplished by adding an inverter chip to the top of the output latch.

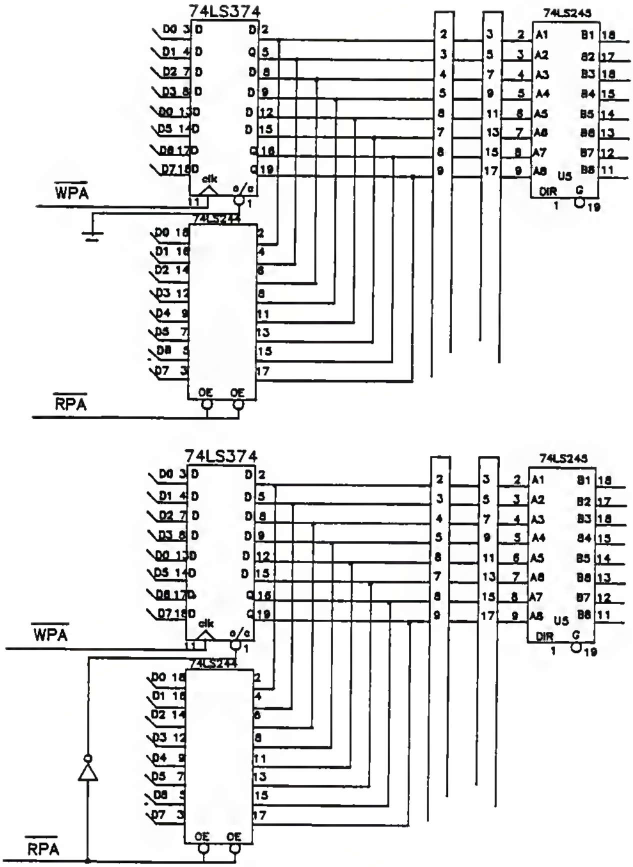

Figure 40. Z-158 Parallel Port Modification.

DACI External Clock Switch Addition

In order to run the EHSI system at variable speeds, a switch was added to the DACI that gives the user the choice of using the system clock or adding an external signal generator. The line connections changed are seen in Fig. 41. The benefit of this modification was apparent in the filtering section, where greater than 2 Hz updates are required. The system can also be pushed to its limit by adjusting the frequency of the signal generator, thus finding the maximum update rate that the system can keep up with.

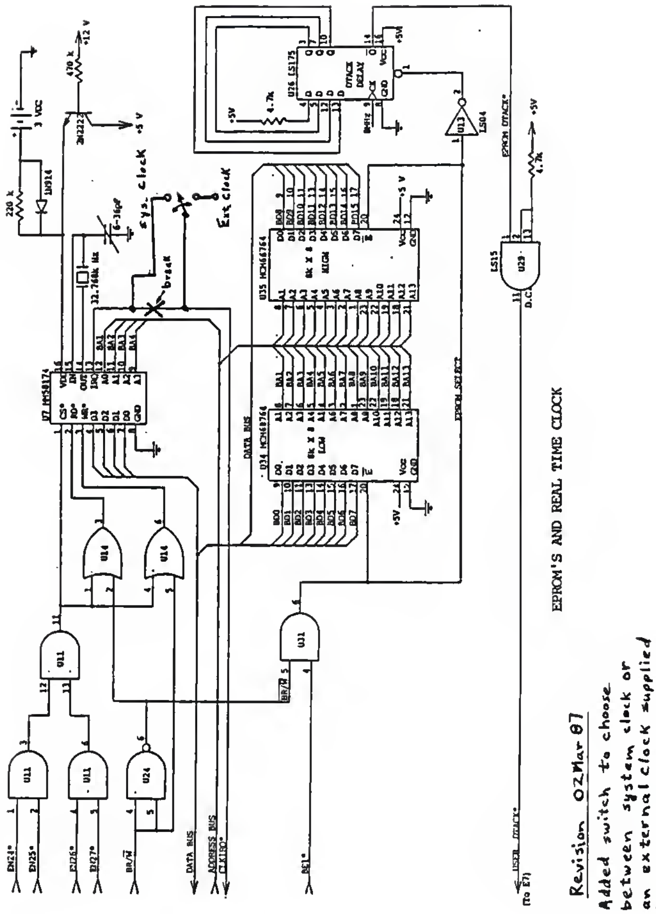

Figure 41. DACI Clock IRQ Switch Addition.

DACI IRQOUT and IRQIN lines addition

Two lines were added to the DACI that helped seperate interrupt requests and handshaking sequences. Figs. 42 and 43 show the change of connections made on the DACI. A routine was added to the DACI software that pulses the DACI IRQOUT line. Dedicated interrupt request and receipt lines are necessary to alleviate timing problems. Interrupts were being missed, and OUTSHAKE pulses were being mistaken for interrupts before the IRQ lines were added. The modification has greatly reduced the number of acknowledge errors and bad interface commands between the Z-158 and the DACI.

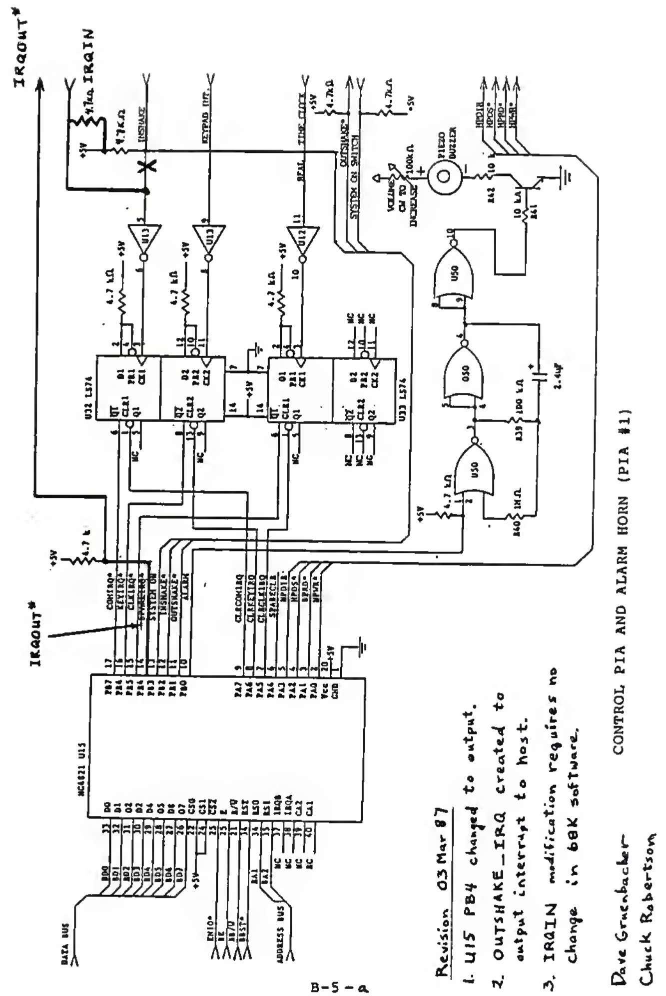

Figure 42. DACI IRQ Lines Addition

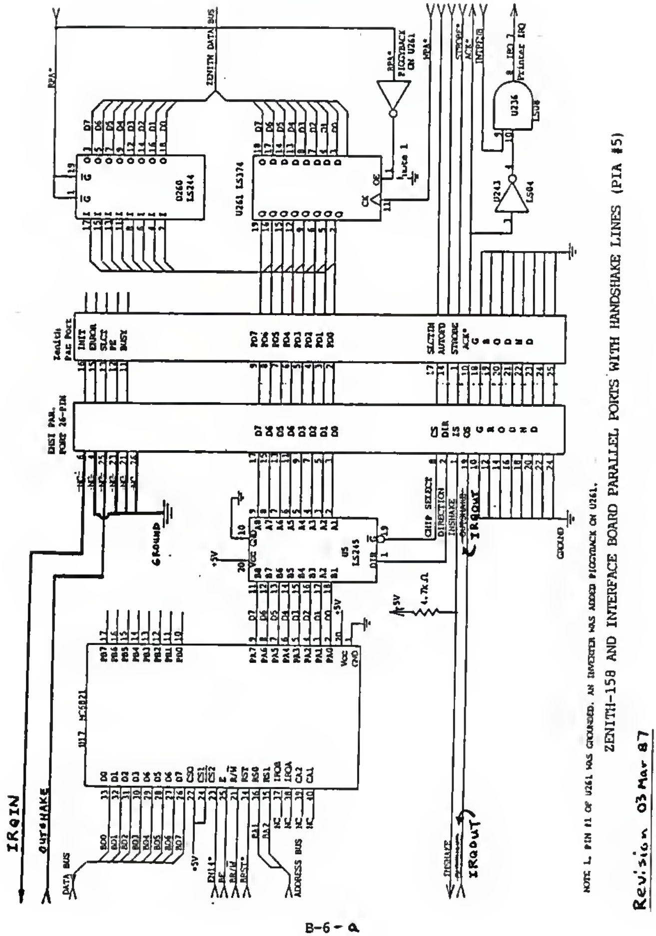

Figure 43. DACI IRQ Lines Addition

LOW-LEVEL SOFTWARE FOR AN EHSI DEVELOPMENT SYSTEM

by

DAVE GRUENBACHER

B.S., Kansas State University, 1985

AN ABSTRACT OF A MASTER'S THESIS

Submitted in partial fulfillment of the
requirements for the degree

MASTER OF SCIENCE

Department of Electrical Engineering

KANSAS STATE UNIVERSITY
Manhattan, Kansas

1987

Assembly-language routines are presented that establish communications between the host computer (Zenith-158), and the smart interface of an electronic horizontal situation indicator (EHSI) development system. Communications are accomplished in an interrupt environment supported by a handshaking protocol. The main program, written in C, is presented as the controlling program for the EHSI system. The main program and communication routines are interfaced together. Functions are presented that service key presses of the system keypad. Flight simulator data is sampled and differentiated, via digital filtering, to extract airplane trend information. User guidelines and maintenance information for the communication routines are given.